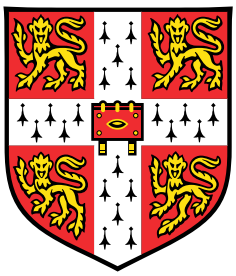
Automated CAD model generation for structural optimisation



Ge Yin

Supervisor: Dr. Fehmi Cirak

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Churchill College October 2019

Declaration

I hereby declare this thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Ge Yin
October 2019

Abstract

Computer-aided design (CAD) models play a crucial role in the design, manufacturing and maintenance of products. Therefore, the mesh-based finite element descriptions common in structural optimisation must be first translated into CAD models. Currently, this translation either can be performed semi-manually or fails to reserve the structural optimality found by the structural optimisation due to the intrinsic difference in geometric representation between finite element mesh and CAD model.

This thesis propose a fully automated and topologically accurate approach to synthesise structurally sound parametric CAD models from topology-optimised finite element models to fill the long-existing gap between structural optimisation and CAD systems. This approach successfully preserves the optimal structural performance during the mesh-CAD conversion.

The solution provided in this thesis is to first convert the topology-optimised structure into a spatial frame structure and then to regenerate it in a CAD system using standard constructive solid geometry (CSG) operations. The obtained parametric CAD models are compact, that is, have as few as possible geometric parameters, which makes them ideal for editing and further processing within a CAD system. The critical task of converting the topology-optimised structure into an optimal spatial frame structure is accomplished in several steps. The first step is to generate a one-voxel-wide voxel chain model from the topology-optimised voxel model using a topology-preserving skeletonisation algorithm from digital topology. The undirected graph defined by the voxel chain model yields a spatial frame structure after processing it with the proposed graph algorithms. Subsequently, the cross-sections and layout of the frame members are optimised to recover its optimality, which may have been compromised during the conversion process. At last, the obtained frame structure is generated in a CAD system by repeatedly combining primitive solids, like cylinders and spheres, using boolean operations. The resulting solid model is a boundary representation (B-Rep) consisting of trimmed non-uniform rational B-spline (NURBS) curves and surfaces.

The numerical studies in this thesis clarify that the converted spatial frame structures are with equivalent structural performance. Moreover, CAD models generated from the spatial frame structures have significantly fewer geometric degree of freedom compared to the topology-optimised structures. Though the numerical studies use topology-optimised structures as input and compact CSG models as output, this thesis also provides the way to extend the proposed generation process to taking other optimised meshes and producing outputs of various geometric representations. This offers a wide range of possible applications and brings new thoughts to industrial design and manufacturing.

***Thesis Title*:** *Automated CAD model generation for structural optimisation*
***Author*:** *Ge Yin*
***Keywords*:** *Topology Optimisation, Computer-Aided Design, Mesh Reconstruction, Digital Topology, Homotopic Skeletonisation, CSG tree*

To My Dear Parents:
Ping Zhang and Yanping Yin

Acknowledgements

First and foremost I would like to express my sincere gratitude to Dr. Fehmi Cirak, my supervisor, for his patient guidance and enthusiastic encouragement to this research work. His expertise in mathematics and computational mechanics are extraordinarily invaluable in formulating the frame work of this research.

I would also like to thank Dr. Malcolm Sabin. Without his insight in computational geometry and his advice in writing, I would not have completed this thesis. My sincere thanks also go to Dr. Xiao Xiao, my colleague and collaborator in this research work, for his important efforts made for this work.

I appreciate the extraordinary morale support and language guidance from Sumudu Herath, and cheers for all the fun we have together. Thanks go to Kim Jie Koh, who proofread my thesis with extreme care and patience, and Tianyi Liu, who helped with the printing during the thesis correction. I am also grateful to all my colleagues in Cambridge CSMLab, Eky Febrianto, Adeeb Arif Kor and Dr. Qiaoling Zhang, for the productive research discussions.

I give my thanks to my friends, Paul Sabota, Kentaro Suda and his wife Suzuka Nakazawa, Dr. Nikita Hari, ShengChi Liu, Tim Watson, Dr. Bing Wang, Linyuan Xin, Saite Lu, Andy Tao and Henry Zhang, for our wonderful memories in Cambridge and for the forever friendship. Especially, to Tim Watson, thank you so much for your brilliant advice in writing.

Finally, The research work in the University of Cambridge is funded by Chinese Scholarship Council and I thank the scholarship committee for offering me the opportunity.

Thanks again to all of you,
Ge Yin

Table of contents

List of figures

List of figures

List of tables

List of tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Motivation

Structural optimisation has become an important tool in the industrial design for the past few decades. It allows to find geometries with minimal stress, weight or compliance for a given amount of material and constraints. These optimised geometries are usually the starting point for further detailed design. Therefore, they need to be first converted into compact Computer-Aided Design (CAD) models. This is necessary because CAD systems are today an integral part of most industrial product development processes. The prevalent parametric CAD systems are based on *Boundary Representation* (B-rep)[1] techniques and trimmed *Non-Uniform Rational B-splines* (NURBSs)[2], which are constructed using *Constructive Solid Geometry* (CSG) trees[3]. These geometric representations are dramatically different from the finite element meshes used in structural optimisation.

Although the input to most structural optimisation is a CAD geometry, its output is usually a finite element mesh of the optimised geometry. The finite element meshes consist of too many elements to be suitable for processing and editing in a CAD system. For a geometry to be editable in a CAD system, a compact representation in the form of a CSG tree is essential. The need to edit the optimised geometry

[1]B-rep is a method for representing solid as a collection of connected surface elements, the boundary between solid and non-solid [1].

[2]NURBS, constructed by combination of polynomial basis functions, is a mathematical model commonly used in computer graphics for generating and representing curves and surfaces. Details of NURBS are clearly written in [2].

[3]CSG tree is a technique commonly used in CAD modelling. It allows a modeller to create a complex surface or solid object by using Boolean operators, e.g. union, subtraction, etc., to combine simpler objects [3].

arises when, for instance, additional geometric features have to be added or when the part is to be combined with other components into a functional product. Moreover, often in industrial practice many not explicitly quantified design requirements, in addition to structural efficiency and robustness, require the geometry to be edited after optimisation.

There are only a very limited number of structural optimisation approaches which arrive at a geometry in the form of a compact parametric CAD model. The difficulties in the mesh-CAD conversion mainly come from the intrinsic difference in geometric representation between finite element models and CAD models. As illustrated in Figure 1.1, in order to achieve an integrated smooth industrial design process, the key challenge is how to convert the optimised shape to a compact CAD model in an automated and robust way. Unfortunately, current solutions proposed in research literature and techniques in engineering software are inefficient and not sufficiently robust and automated, as to be discussed in the next section. Thus, converting mesh-based optimised geometries to compact CAD models demands tedious manual editing and enormous time during industrial design. This strongly necessitates the research on robust as well as automated methods to fill the gap between structural optimisation software and CAD systems.

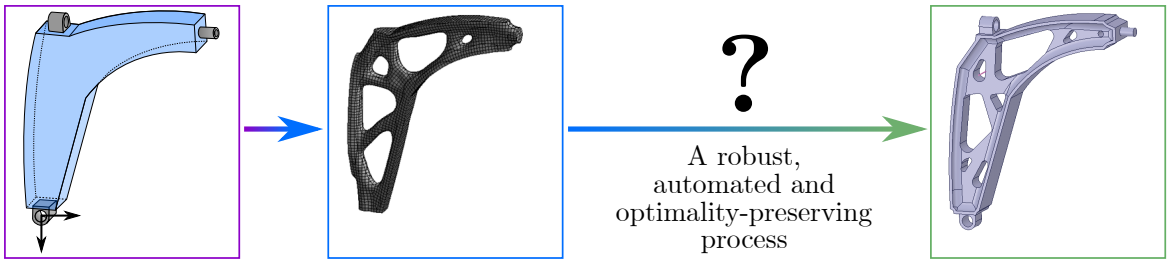


Fig. 1.1 Missing link between structural optimisation and CAD systems. The ideal workflow is, from left to right, receiving design input, then getting optimised structure and finally outputting compact CAD model.

1.2 Background

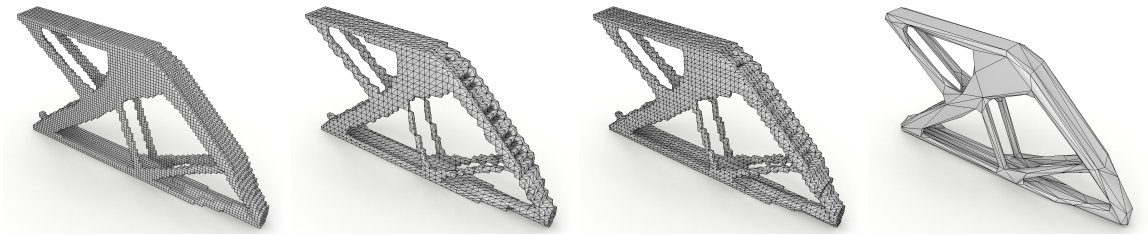
Structural optimisation is the subject of making an assemblage of materials sustain loads in the best way [4]. Dependant on which optimisation variables are used, structural optimisation can be presented into three types: (i) sizing optimisation, e.g. optimising the cross-sectional area of a beam, (ii) shape optimisation, e.g. optimising shape of

a thin-shell, and (iii) topology optimisation that optimises the material distribution in the design domain. Among these three, topology optimisation is the most general form of structural optimisation; with proper problem definitions, topology optimisation can also solve sizing and shape optimisation problems [4]. All structural optimisation techniques are based on the iterative finite element analysis.

However, the inherent differences between the geometric representations used in CAD systems and conventional finite element analysis induce compatibility issues when reimporting optimised results to CAD systems [5]. There are efforts made in directly using CAD geometric representations for structural optimisation. Such approaches are primarily restricted to shape optimisation. For instance, it is quite common to use the parameters of a CAD or a reconstructed CAD-like spline model as geometric design variables in shape optimisation, see e.g. [6–9]. These techniques are especially appealing in an isogeometric analysis context when the same basis functions are used for geometry description and finite element analysis [10–15]. Several research [16–18] reveal the possibility to combine the topology optimisation and isogeometric shape optimisation to achieve the isogeometric topology optimisation. However, these methods are fundamentally challenged by the discontinuous nature of changing the topology in a mechanical model. Moreover, the crisp description of the structural boundaries used in these methods may lead to an abrupt change of the structural behaviour as geometric features merge or separate [19]. As a result, the topology optimisation based on conventional finite element analysis are currently the most widely accepted one by the research community as well as the industry. Since most structural optimisation techniques cannot directly produce CAD-compatible results, one promising way is to reconstruct optimised meshes as parametric CAD models.

Reconstruction of a CAD model from a mesh is still an open problem in engineering and computational geometry. It is well understood that directly applying common surface mesh smoothing and simplification algorithms do not yield compact geometric representations. This is because most mesh smoothing and simplification algorithms rely heavily on the positions of mesh nodes and the orientations of mesh patches, e.g. *Laplacian smoothing* [20] smooths a mesh by averaging the coordinates of the nodes within a given region and the cutting results of *marching cube* [21] grids majorly depend on the normal directions of mesh patches. As a result, these algorithms do not function well on a jagged mesh. For example, the volume mesh in Figure 1.2a is a topology-optimised cantilever that is commonly seen in topology optimisation literature [22]. The results of using aforementioned algorithms, shown in Figures 1.2b and 1.2c, still contain large numbers of elements, and their boundary surfaces are not sufficiently

smooth. The mesh in Figure 1.2d shows better compactness and smoothness than its two previous counterparts, but the edition on this mesh is far from trivial, e.g. even increasing the size of the bottom plate requires numerous manual operations.



(a) (b) (c) (d)

Fig. 1.2 Volume mesh is simplified using common mesh algorithms. (a) is the input mesh. Directly applying cluster decimation [23] on (a) gives (b). (c) is obtained by using cluster decimation on Laplacian smoothed (a). (d) comes from smoothing (a) using the marching cube method then gets simplified using quadratic edge collapse decimation [23].

Current solutions to tackle the conversion problem of structural-optimised meshes to compact CAD models are grouped into two major types, namely the spline patches approach and the medial axis approach.

1.2.1 The spline patches approach

The first group creates a compact spline representation of the optimised geometry to be imported into a CAD system. The spline representation reconstruction methods can be referred to as reverse engineering of existing physical objects. Such methods can reconstruct CAD models from the acquired geometry and geometric features representing the geometry of a physical part, see the comprehensive reviews [24, 25].

Some work of this approach focuses on reconstructing compact spline surfaces from meshes. In particular, meshes are first partitioned using, for instance, *watershed* segmentation proposed by Mangan and Whitaker [26] (based on local surface curvature minima) or *hierarchical clustering* by Garland et al. [27] (clustering method with a bias on surface normal directions). From the partitioned mesh, a more compact mesh can be generated by approximating each partition with a coarse mesh using, e.g. variational shape approximation by Cohen-Steiner et al. [28], quadric surface fitting by Yan et al. [29], B-spline fitting by Eck and Hoppe [30], or the implicit surface reconstruction by Rouhani et al. [31]. Evidently, the surface fitting is performed on individual partitions, since fitting surfaces for the whole mesh is not possible for

structural and mechanical geometries with non-trivial topologies [32]. However, it is worth emphasising that these methods still require significant manual interactions. The very recent automated method developed by Marinov et al. [32] achieves a smooth and compact reconstruction using B-spline patch fitting and contact boundary smoothing starting from an already partitioned mesh. Though their method changes the shape and geometric representation (solid to surface), they did not consider whether the optimality in mechanical performance of the generated model is preserved.

For the non-smooth meshes, which are commonly obtained from topology optimisation, most partitioning methods will not function well due to the sudden changes in the boundary face normal vectors. In order to make use of existing CAD model reconstruction techniques, a common approach is to obtain sufficiently smooth surface meshes from topology-optimised shapes using *density isocontours*; density isocontours are used to segment voxels according to the optimal element densities determined with topology optimisation. Spline curves in two dimensions (2D) or spline surfaces in three dimensions (3D) [33–35] are subsequently fitted to the segmented mesh parts. However, these techniques are not robust, especially in 3D, as they may require the solution of a non-linear least squares problem and are difficult to automate due to the need to manually position the control vertices of the splines. As a result, they have had no noteworthy impact on commercial software despite being developed around two decades ago.

1.2.2 The medial axis approach

The second group of methods is to use *medial axis* of mesh-based models to construct compact CAD models. The essential component in these methods is *skeletonisation*m which is applied to obtain the medial axis. Skeletonisation is an active research area with applications in computer graphics, animation and volumetric image processing, see the reviews [36–39]. The medial axis, also referred to as curve skeleton, of a 3D object is closely related to its medial surface, or surface skeleton. In finite elements, medial surfaces are known from mesh generation applications, see e.g. [40]. Informally, the medial surface is the set of all points that have two or more closest points on a 3D object's surface. That is, a sphere centred on the medial surface touches the object's surface at two or more points. In contrast to the medial surface, the skeleton of a 3D object has no rigorous definition. It is expected that the skeleton captures the essential topology of the object and is centred, i.e. lies on or close to the medial surface. There are many algorithms for determining the skeleton of an object starting

from different types of geometry representations, such as implicit signed distances, or polygonal surfaces etc.

In the context of structural engineering, mesh reconstruction based on medial axes has been originally pioneered in Bremicker et al. [41] for 2D. The skeletonisation technique used in their methods is proposed by Arcelli et al. [42], which is the early stage of the so-called *thinning algorithm*. Thinning algorithm specialises in efficient and robust skeletonising volume meshes. The mathematical basis of thinning algorithm is *digital topology*, which has been extensively researched in the few decades [39, 43–45]. In topology optimisation, it is expedient to assume that the object is given as a voxelised binary image consisting of solid and void voxels. Digital topology provides a rigorous basis to study the topology of binary images [46]. Crucially, the entire skeletonisation process does not rely on any floating point operations, which makes the thinning algorithm exceedingly robust. These make the thinning algorithm the best choice in skeletonising topology-optimised meshes. While in [41] the aim is not to arrive at a parametric CAD model, the thinning algorithm used to extract a pin-jointed truss structure, which is subsequently size and layout optimised. Homotopic skeletonisation in 3D is substantially more challenging than in 2D, and elimination of possible mechanisms in a spatial truss structure is far from trivial [47]. And so, in this thesis, a skeleton obtained from thinning algorithm is converted into a spatial frame structure (with rigidly connected joints) that intrinsically does not exhibit mechanisms. And so, in this thesis, the skeleton is converted into a spatial frame structure (with rigidly connected joints) that intrinsically does not exhibit mechanisms.

Very recently, Cuillière et al. [48] used a skeleton extraction method based on surface mesh collapse proposed by Au et al. [49] to extract a curve skeleton from the surface mesh representing the density isocontour of the optimised geometry. This specific extraction technique is rather elaborate as it builds on the successive contraction of a given surface mesh using Laplacian smoothing. The obtained skeleton and estimated cross-sections are used to generate a CAD model with no further post-processing. However, besides, Cuillière et al. do not account for the fact that skeletonisation may have impaired the optimality of the structure.

1.3 Contributions

The primary contribution of this thesis is a novel, versatile and robust compact CAD model generation process for structural optimisation applications. This generation process is fully automated and can robustly generate a compact CSG tree representation

of topology-optimised geometries and convert them to CAD models. The process establishes an important link between finite element mesh and compact CAD model and solves the long-existing incompatibility issues between structural optimisation software and CAD systems. Importantly, this process preserves the optimality discovered by topology optimisation, while other existing methods do not consider the optimality in structural performance after the mesh-CAD conversion [32, 41, 48, 50]. Furthermore, this process can be expanded to taking other optimised mesh-based geometry inputs and producing outputs of various geometric representations in a straightforward way. An overview of the approach is provided in the next section.

Other contributions of this thesis are as follows:

- Validate the robustness of the proposed sequential frame size and layout optimisation proposed in this thesis in the context of CAD model generation. The enforcement of geometric constraints using implicit representations enables the frame optimisation on complex domains. This ensures the compatibility of the frame optimisation process with other structural optimisation methods.
- Introduce techniques for maintaining user-defined features during the skeletonisation process. These are necessary to apply the skeletonisation algorithm in engineering design problems. Skeletonisation on geometries with prescribed features has been tested, and the resulting skeletons of those models show that both curve and surface skeleton will grow to reach user-defined features in a homotopic way.
- Provide algorithms for extracting a graph model from a voxel chain model. The connectivity of the graph model can be conveniently adjusted using graph algorithms to fit structural and manufacturing requirements. Several tools are developed for post-processing the graph model into a well-conditioned frame model.

Until the submission of this thesis, the research work of this thesis has been summarised as a preprint [51] and presented in Conference of UK Association of Computational Mechanics 2019 (UKACM2019) and 15th U.S. National Congress on Computational Mechanics (USNCCM15).

1.4 Overview of the proposed approach

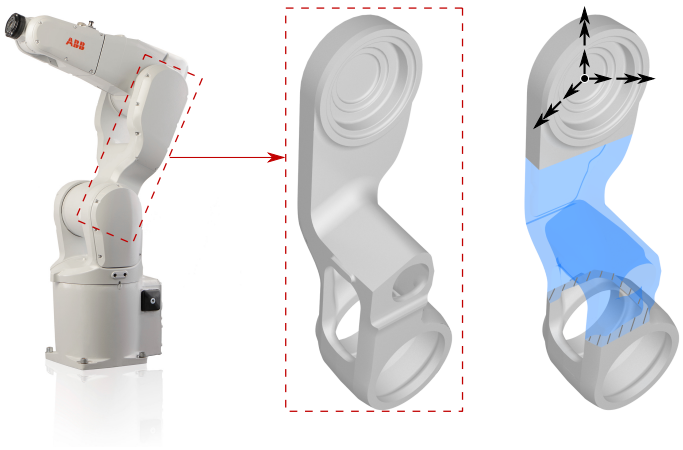
For clarity, this section presents an overview of the proposed robust, automated and optimality-preserving process for converting meshes into compact CAD models for

structural optimisation applications. As an example, the complete workflow of the topology optimisation of a robot arm is illustrated in Figure 1.3. The input is a CAD model (e.g. a B-rep consisting of trimmed NURBS surfaces) along with design requirements (loadings, boundary conditions, volume reduction), and the output is a structurally optimised shape in the form of a compact parametric CAD representation.

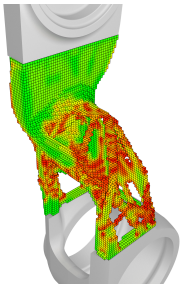
Here topology optimisation uses a standard density-based modified *Solid Isotropic Material with Penalisation* (SIMP) approach[1] on a structured hexahedral finite element mesh, referred to as a voxel mesh, see e.g. [22, 55, 56]. The optimised density field, as shown in Figure 1.3b, splits the set of voxels, after thresholding into the two subsets solid and void, which gives a 3D binary picture with two grey levels. Subsequently, from the binary image a voxel chain skeleton is extracted, see Figure 1.3c, using a homotopic, i.e. topology-preserving, skeletonisation algorithm from digital topology [44]. The skeleton has the same number of connected components, handles and cavities as the three-dimensional voxel model. Evidently, topology preservation is critical in structural applications because a change in topology may disrupt the load paths discovered during optimisation. The voxel chain model defines a weighted undirected graph which is processed with graph algorithms to obtain a spatial frame structure, see Figure 1.3d. Each of the members of the frame structure is a beam and the beams are rigidly connected at the joints. During the conversion of the voxel model to the spatial frame structure, the optimality of the structure is usually compromised. However, this can be recovered by applying a few steps of size and layout optimisation to the spatial frame structure. Size optimisation adjusts the cross-sections of the members and layout optimisation adjusts the coordinates of the joints. The very compact representation that is generated from the original voxel model consists of the connectivity of the frame structure, its joint coordinates and the member cross-sections. These are used to create a binary CSG tree and a solid model of the spatial frame structure in a CAD system. Figures 1.3e and 1.3f show two different solid models generated in a CAD system. In the first model, the cylindrical members are connected to spherical joints, while in the second model, the members are smoothly blended at the joints. With the CAD model generation process presented in this thesis, the first model is generated in a fully automated fashion, while the second requires some user intervention[2].

[1] Although here the approach uses a density-based approach for topology optimisation, it is straightforward to apply the proposed approach to the more recent level-set based methods [52, 53] or the historical homogenisation method [54].

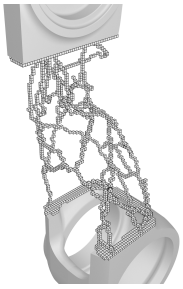
[2] Here surfaces are first connected using blend surfaces then the NURBS patch edges are smoothed using the edge fillet. These operations are available in most CAD software. But they require carefully choosing operation parameters, which are completely shape-dependant. Thus it is not trivial to automate these operations.



(a) Design space and loading

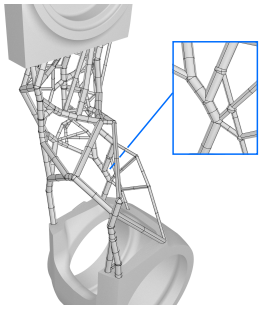


(b) Topology-optimised geometry

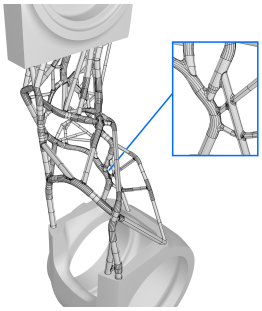


(c) Homotopically thinned voxel chain model

(d) Spatial frame model



(e) CAD model generated by recursively combining primitive solids



(f) CAD model generated by combining primitive solids and blended surfaces

Fig. 1.3 CAD model generation workflow for a topology-optimised robot arm. The robot arm is a lever mechanism which pivots around the bottom cylindrical hole shown in (a) when forces and torsions are applied at the top. Due to non-structural constraints, only the middle part can be optimised. The topology-optimised geometry (b) is skeletonised into voxel chain (c), then the voxel chain is converted in to a spatial frame model. Finally, in (e) and (f), the optimised frame has been merged with top and bottom parts in a CAD system.

1.5 Outline

The outline of this thesis is as follows.

Chapter 2 introduces the theoretical basis for topology optimisation and structural frame optimisation. The chapter first reviews the different formulations with a particular focus on the density-based approach of topology optimisation, followed by the sensitivity analysis used for the gradient-based optimisation algorithm. Next, the structural frame optimisation is discussed with the formulation and its derivatives. This is followed by a discussion on the enforcement of implicitly defined geometric constraints so that the structure remains within a given design domain.

Chapter 3 explains the relevant aspects of digital topology and provides the technical details of the skeletonisation algorithm. The topological invariant, known as the Euler characteristic, is introduced and explained. Then, the criteria for preserving global and local topology are clarified. Such criteria are fundamental to the homotopic skeletonisation algorithm. The robustness and efficiency of the skeletonisation algorithm are studied using several skeletonisation examples.

Chapter 4 covers the post-processing of the obtained medial axis given in the form of a voxel chain. Details are given on extracting an weighted undirected graph model from a voxel chain model. Subsequently, the rules for manipulating graph models using incidence matrices are provided. With the frame model at hand, the chapter considers two different types of CAD models as the output options, i.e. subdivision surfaces and NURBS-based B-rep model. Then the methods to generate compact CAD models in either of these two forms are elaborated.

Chapter 5 presents several numerical studies of the proposed approach. At the beginning, the entire workflow consisting of optimisation and geometry conversion processes is summarised. Next, the proposed CAD model generation process is used to various examples ranging from a standard topology optimisation benchmark example to complex geometries. For these examples, studies are conducted in particular on the change in cost function during the conversion of the optimised voxel model to a CAD model to demonstrate that the proposed approach is optimality-preserving.

Chapter 6 concludes the thesis with a summary of the presented topologically robust CAD model generation approach and provides an outlook for future research.

CHAPTER 2

Review of structural optimisation

Structural optimisation offers improvement of structural design by reducing material usage, achieving better performance and shortening manufacturing time [57]. Chapter,covers the three commonly used structural optimisation methods: (i) topology optimisation, (ii) frame size optimisation and (iii) frame layout optimisation. First, a brief tour is given of the underlying formulations and relevant numerical techniques of topology optimisation in Section 2.1. Next, Section 2.2 provides the fundamentals of frame size and layout optimisation. Then Section 2.2.4 gives a new idea of using implicit geometric representation as frame optimisation constraints. Several numerical examples for frame size and layout optimisation are presented in Section 2.2.5. The limitations of frame optimisation, specifically layout optimisation, are discussed in Section 2.2.6.

2.1 Review of Topology optimisation

Topology optimisation originates from the field of structural optimisation and optimises the material distribution within a design domain for a given set of loadings and boundary conditions [58]. Topology optimisation aids the development of new products by finding the best possible layouts in the conceptual design stage. Topology optimisation techniques were first developed by Bendsøe and Kikuchi [54], who resorted to artificial material densities to solve structural optimisation problems. Later this concept was extended by Sigmund [59] to optimising compliant structures consisting of multiple materials. Comprehensive reviews can be found in [60–64].

In topology optimisation the *density* is the optimisation variable, which represents the amount of material at each point in the design domain. Topology optimisation was initially considered as a binary optimisation problem with the density of 1 and 0 representing solid and void, respectively. Discrete search for solving such binary problem is so expensive (indeed a NP hard problem) [65] that current approaches optimise the continuous problem, such as the *homogenisation approach* [54, 66–68], *level-set approach* [53, 69] and the *density-based approach* [56]. This thesis uses the density-based approach primarily because of its computational simplicity, but also because the homogenisation approach exhibits poor manufacturability whereas the level set approach requires expensive sensitivity analysis [65].

The density-based approach is an *Eulerian approach* whereby the design domain is discretised into finite elements [65]. The density-based approach is also a continuous approach employing the *Solid Isotropic Material with Penalisation* (SIMP) method [56]. However, representing the density continuously means that intermediate density values can exist. In order to achieve a close-to-binary solution (i.e. solid and void), the Young's modulus of each finite element is penalised according to a power-law density. The *modified SIMP* method [70] introduces a small non-zero number as the Young's modulus for void elements to ease the singularity caused by zero Young's modulus.

The density-based approach is known to exhibit numerical issues, namely *checkerboarding* and *mesh-dependency* [71]. Checkerboarding refers to the periodic-like pattern of solid and void elements, arranged in a fashion of a checkerboard [71–73]. Meanwhile, mesh-dependency refers to the dependence of the optimisation results on the finite element mesh. Evidently, both checkerboarding and mesh-dependency are undesirable from a practical viewpoint. Therefore, regularisation techniques have been introduced, such as sensitivity filtering [73] and density filtering [74]. As the names suggest, they filter, i.e. smooth, either element sensitivities or densities. It has been shown that both filters suppress checkboarding and produce mesh-independent designs [22]. In general, the optimal shapes given by these two filters are slightly different. In this thesis, density filtering is used.

Gradient-based optimisers are commonly used in solving topology optimisation problem due to their high efficiency, while non-gradient methods become prohibitively expensive for large systems [65]. There are three major gradient-based algorithms: *Sequential Quadratic Programming* (SQP) [75], *Method of Moving Asymptotes* (MMA) [76] and *Optimality Criteria* (OC) method [70, 77]. Compared to the SQP and MMA, the OC method stands out with its extreme efficiency and ease of implementation [4, 78]. In the numerical experiments [22, 77] involving models with large number of

degrees of freedom, the OC method shows better convergence rate than other two methods. For the interested reader, Appendices C.1 and C.2 give the introduction on the SQP and MMA, respectively.

2.1.1 Optimisation formulation

The topology optimisation problem for finite element discretised solids reads

$$\min_{\mathbf{0}\leq\boldsymbol{\rho}\leq\mathbf{1}} \quad J(\boldsymbol{\rho}) = \mathbf{f}^{\mathsf{T}}\,\mathbf{u} = \mathbf{u}^{\mathsf{T}}\,\mathbf{K}\,\mathbf{u} \tag{2.1a}$$

$$\text{subject to} \quad \mathbf{K}\,\mathbf{u} = \mathbf{f} \tag{2.1b}$$

$$\frac{V(\boldsymbol{\rho})}{\overline{V}} \leq V_f\,, \tag{2.1c}$$

where $J(\boldsymbol{\rho})$ is the compliance cost function, $\boldsymbol{\rho}$ is the vector of relative element densities, $\mathbf{u}$ is the global displacement vector, $\mathbf{K}$ is the global stiffness matrix, $\mathbf{f}$ is the global external force vector, $V(\boldsymbol{\rho})$ is the material volume, $\overline{V}$ is the design domain volume and V_f is the scalar prescribed volume fraction. In this thesis, the design domain is always discretised with a structured grid and hexahedral linear elements. As usual, the global stiffness matrix $\mathbf{K}$, vector $\mathbf{u}$ and vector $\mathbf{f}$ are assembled from the n_{ele} element contributions $\mathbf{K}_i \in \mathbb{R}^{24\times24}$, $\mathbf{u}_i \in \mathbb{R}^{24\times1}$ and $\mathbf{f}_i \in \mathbb{R}^{24\times1}$ in the mesh respectively, i.e. $\mathbf{K} = \mathcal{A}_{i=1}^{n_{\text{ele}}}\,\mathbf{K}_i$, $\mathbf{u} = \mathcal{A}_{i=1}^{n_{\text{ele}}}\,\mathbf{u}_i$ and $\mathbf{f} = \mathcal{A}_{i=1}^{n_{\text{ele}}}\,\mathbf{f}_i$. The relative density of each element is constrained to be $0 \leq \rho_i \leq 1$ with $i = 1, 2, \ldots, n_{\text{ele}}$, where n_{ele} is the number of elements. The boundary conditions are applied using *Lagrange multipliers* [79].

2.1.2 Modified SIMP method

In each element i the material is isotropic and homogeneous, and the Young's modulus E is penalised depending on the relative density ρ_i with

$$E(\rho_i) = E_{\min} + \rho_i^p(\overline{E} - E_{\min})\,, \tag{2.2}$$

where $\overline{E}$ is the prescribed Young's modulus of the solid material, p and $E_{\min}$ are two algorithmic parameters. When no penalisation power is applied, i.e. $p = 1$, there is a large fraction of elements with intermediate density values, i.e. $0 < \rho_i < 1$; such elements can be seen in the yellow region in Figure 2.2a. These intermediate densities describe a structure which is very different from the sought layout with only solid and void elements. Once the penalisation power is introduced, see Figure 2.1, the contrast between the solid elements and void elements becomes distinct. As illustrated

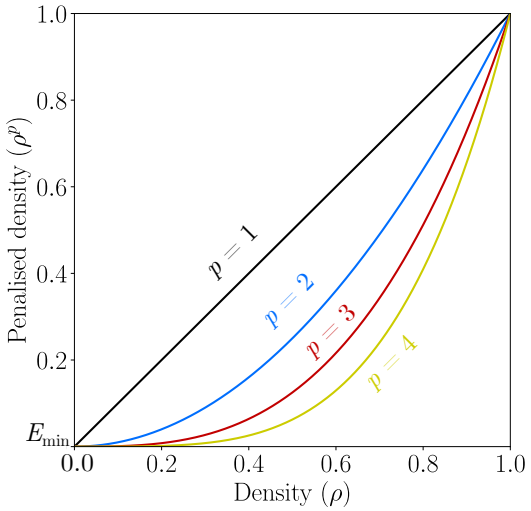
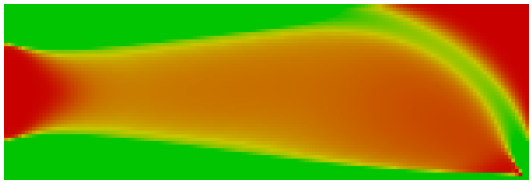
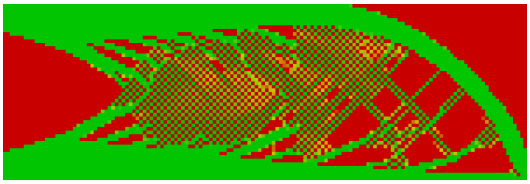


Fig. 2.1 Penalised density. As the penalisation increases, the penalised density ρ^p becomes closer to either $\rho = 1$ or $\rho = 0$.



(a) Densities are penalised with $p = 1$, minimum reached after 33 iterations, $J(\hat{\boldsymbol{\rho}}) = 16.128$.



(b) Densities are penalised with $p = 2$, minimum reached after 143 iterations, $J(\hat{\boldsymbol{\rho}}) = 17.454$.

(c) Densities are penalised with $p = 3$, minimum reached after 56 iterations, $J(\hat{\boldsymbol{\rho}}) = 18.522$.

(d) Densities are penalised with $p = 4$, minimum reached after 30 iterations, $J(\hat{\boldsymbol{\rho}}) = 20.263$.

Fig. 2.2 Topology optimisation using various penalisation powers p. In the plots green denotes $\rho = 1$ and red $\rho = 0$. The Young's modulus for solid elements is $\overline{E} = 10^5$ and for void elements is $E_{\min} = 10^{-9}$, and Poisson ratio is $\nu = 0.3$; the cantilever is fixed on the left edge and has a downward force of 100 on the right bottom corner. The cost function values of the shape of 150×50 unit elements are shown in the legends. Except for the penalisation power, the rest of optimisation parameters, element sizes and design systems used for these four optimisations are identical. All optimisations here proceed without any filters and the volume fraction $V_f = 0.5$.

in Figures 2.2b to 2.2d, in contrast to Figure 2.2a, the intermediate densities are no longer observed, and instead, the solid elements (in green) can be easily extracted from the optimised layout. Along with the increase of penalisation power the optimality of the optimised layout drops. For example, the cost function value of the optimised layout in Figure 2.2a is less than its three counterparts in Figures 2.2b to 2.2d. This is a trade-off between optimality and manufacturability. As suggested in [56], the penalisation parameter $p \geq 3$ ensures that elements with densities close to $\rho_i = 0$ (void) and $\rho_i = 1$ (solid) are preferred. Besides, the small Young's modulus $E_{\min} \approx 10^{-9}$ of the void material prevents ill-conditioning of the global stiffness matrix when $\rho_i = 0$.

Let $\overline{\mathbf{K}}_i \in \mathbb{R}^{24\times 24}$ denote the element stiffness matrix corresponding to the prescribed Young's modulus of the solid material $\overline{E}$. During topology optimisation, $\overline{\mathbf{K}}_i$ remains constant, so it is commonly precomputed and stored in the process. Accordingly, the element stiffness matrix $\mathbf{K}_i$ can be expressed using $\overline{\mathbf{K}}_i$ as

$$\mathbf{K}_i(\rho_i) = \frac{E(\rho_i)}{\overline{E}}\overline{\mathbf{K}}_i\,. \tag{2.3}$$

Details of solid element stiffness matrix $\overline{\mathbf{K}}_i$ can be found in Appendix A.1.

2.1.3 Density filtering

After the introduction of the penalisation power p, the optimised layout is still far from satisfactory. For instance, the checkerboarding is observed in the optimised geometry. In As can be seen in Figures 2.2b, 2.2c, and 2.2d, a large amount of solid elements connected to each other with only one node. The checkboarding commonly occurs in the region that has intermediate densities if a lower penalisation power is applied [56].

In order to prevent checkerboarding instabilities, the solution to the optimisation problem (2.1) can be regularised by filtering the element densities ρ_i [73, 74]. This is accomplished by convolving the element densities with the kernel function

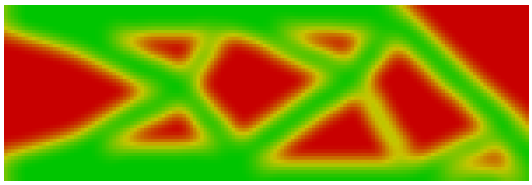
$$H(i,\,j) = \begin{cases} R - \mathrm{dist}(i,\,j) & \text{if } \mathrm{dist}(i,\,j) < R \\ 0 & \text{otherwise}\,, \end{cases} \tag{2.4}$$

where $\mathrm{dist}(i,\,j)$ gives the Euclidean distance between the centroids of elements i and j, and R is the prescribed filter radius. With $H(i,\,j)$, the filtered densities are given by

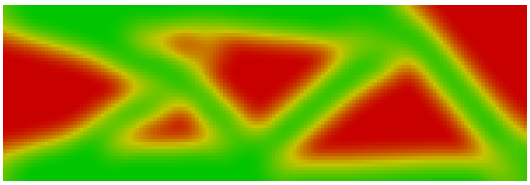
$$\hat{\rho}_i = \frac{\sum_j H(i,\,j)\,\rho_j\,v_j}{\sum_j H(i,\,j)\,v_j}, \tag{2.5}$$

(a) Densities are filtered with $R = 1$, minimum reached after 56 iterations, $J(\hat{\boldsymbol{\rho}}) = 18.522$.

(b) Densities are filtered with $R = 3$, minimum reached after 181 iterations $J(\hat{\boldsymbol{\rho}}) = 20.546$.

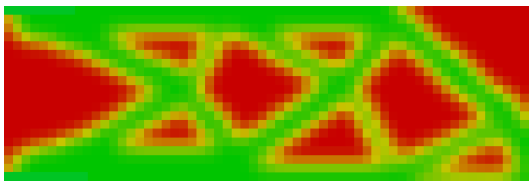


(c) Densities are filtered with $R = 6$, minimum reached after 510 iterations, $J(\hat{\boldsymbol{\rho}}) = 24.688$.

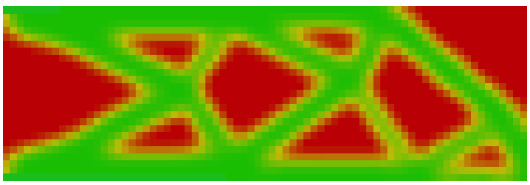


(d) Densities are filtered with $R = 8$, minimum reached after 549 iterations, $J(\hat{\boldsymbol{\rho}}) = 26.691$.

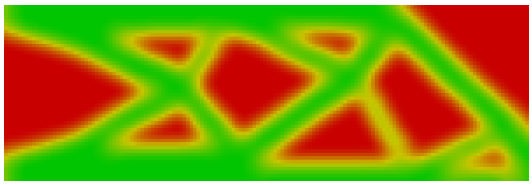
Fig. 2.3 Topology optimisation using various filter radii R. The design problem and material settings are the same as in Figure 2.2. Except for the filter radii, the rest of optimisation parameters and element sizes are identical in the four optimisations. The penalisation parameter is $p = 3$ and the volume fraction is $V_f = 0.5$.



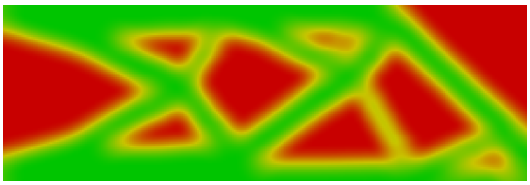
(a) Mesh 60×20, element size 2.5×2.5, minimum reached after 115 iterations, $J(\hat{\boldsymbol{\rho}}) = 24.286$.



(b) Mesh 75×25, element size 2×2, minimum reached after 260 iterations, $J(\hat{\boldsymbol{\rho}}) = 24.360$.



(c) Mesh 150×50, element size 1×1, minimum reached after 510 iterations, $J(\hat{\boldsymbol{\rho}}) = 24.688$.



(d) Mesh 300×100, element size 0.5×0.5, minimum reached after 1824 iterations, $J(\hat{\boldsymbol{\rho}}) = 24.931$.

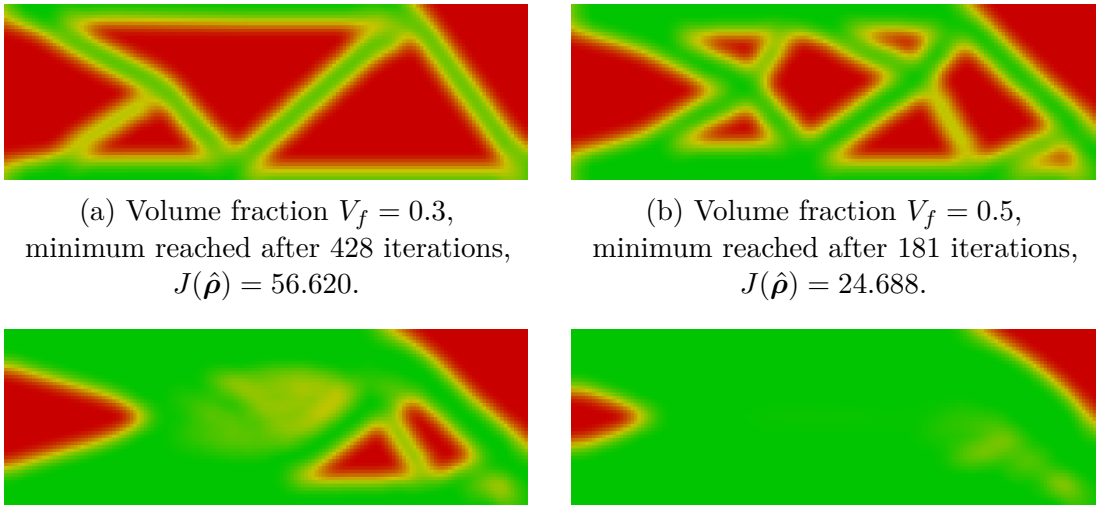
Fig. 2.4 Topology optimisation on various mesh resolutions. The design system and material settings are the same as in Figure 2.2. The design domains for all four are 150×50, and different mesh resolutions use different element sizes. Four optimisations here run using filter radius $R = 6$ and penalisation power $p = 3$.

where v_j is the volume of element j and the summations are over the elements within the filter radius of element i.

As can be seen in Figure 2.3, filtering of the densities significantly changes the optimised layout. For $R > 1$, see Figures 2.3b, 2.3c and 2.3d, the checkerboarding is suppressed. As expected, the higher the filter radius is, the less small features exist in the optimised layout. In general, a wider filter range leads to a higher structural compliance of the final shape, which is not preferred. The filter range is also a trade-off between the structural performance and manufacturability of the optimised layout.

Another purpose of filtering the densities is to reduce the mesh-dependency of the solution. As shown in Figure 2.4, four different mesh resolutions are used to optimise the same design domain. The filter radius is set to 6 for all four optimisations. Visually the optimised layout in Figures 2.4a, 2.4b, 2.4c and 2.4d are all similar to each other. Moreover, the cost functions and structural compliances of the displayed four optimal shapes are very similar up to minor differences.

2.1.4 Choice of volume fraction



(a) Volume fraction $V_f = 0.3$, minimum reached after 428 iterations, $J(\hat{\boldsymbol{\rho}}) = 56.620$.

(b) Volume fraction $V_f = 0.5$, minimum reached after 181 iterations, $J(\hat{\boldsymbol{\rho}}) = 24.688$.

(c) Volume fraction $V_f = 0.7$, minimum reached after 2495 iterations, $J(\hat{\boldsymbol{\rho}}) = 16.062$.

(d) Volume fraction $V_f = 0.9$, minimum reached after 259 iterations, $J(\hat{\boldsymbol{\rho}}) = 13.048$.

Fig. 2.5 Topology optimisation with various prescribed volume fractions V_f. The design system and material settings the same as in Figure 2.2. In all cases the filter radius is $R = 6$ and penalisation power is $p = 3$. The case of $V_f = 0.1$ is not shown here because the optimisation process does not converge.

The choice of volume fraction directly affects how much reduction of material is to be achieved during topology optimisation. As shown in Figure 2.5, it is clear that the topology optimisation process produces qualitatively different optimised layouts for different prescribed volume fractions.

As the volume fraction increases, the optimised layout is more like a plate structure than a frame structure. Note that, very low volume fractions may cause the optimisation process not to converge. For example, for the case in Figure 2.5, the optimisation process cannot proceed if $V_f \leq 0.1$, since there is not sufficient material to establish a distinctive load path from the applied loadings to the boundary. For the user pursuing large volume reductions, reducing the filter radius is recommended.

2.1.5 Sensitivity analysis

For gradient-based optimisation the derivatives of the cost and constraint functions in (2.1) with respect to the relative densities ρ_i are needed. In the following the relative densities ρ_i in the topology optimisation problem (2.1) are replaced with the filtered relative densities $\hat{\rho}_i$. The derivative, or sensitivity, of the compliance cost function (2.1a) according to the chain rule reads

$$\frac{\partial J(\hat{\boldsymbol{\rho}})}{\partial \rho_i} = \frac{\partial J(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} \frac{\partial \hat{\rho}_j}{\partial \rho_i}\,, \tag{2.6}$$

with

$$\frac{\partial J(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} = \mathbf{f}^{\mathsf{T}} \frac{\partial \mathbf{u}(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j}\,. \tag{2.7}$$

Differentiating and rearranging the equilibrium constraint (2.1b) gives

$$\frac{\partial \mathbf{u}(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} \mathbf{u}\,. \tag{2.8}$$

Introducing (2.3) and (2.8) into (2.7) yields

$$\begin{aligned} \frac{\partial J(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} &= -\mathbf{u}^{\mathsf{T}} \frac{\partial \mathbf{K}(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} \mathbf{u} \\ &= -\sum_{l=1}^{n_{\text{ele}}} \mathbf{u}_l^{\mathsf{T}} \frac{\partial \mathbf{K}_l(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j} \mathbf{u}_l \\ &= -p\, \rho_j^{p-1} \frac{\overline{E} - E_{\min}}{\overline{E}} \mathbf{u}_j^{\mathsf{T}} \overline{\mathbf{K}}_j \mathbf{u}_j\,. \end{aligned} \tag{2.9}$$

The derivative of the filtered densities (2.5) is

$$\frac{\partial \hat{\rho}_j}{\partial \rho_i} = \frac{H(j,\, i)\, v_i}{\sum_k H(j,\, k)\, v_k}\,, \tag{2.10}$$

where the summation in the denominator is over the elements within the filter radius of element j. Substituting (2.9) and (2.10) into (2.6) gives the sensitivity of the cost function with respect to the optimisation variable, i.e. unfiltered density, as

$$\frac{\partial J(\hat{\boldsymbol{\rho}})}{\partial \rho_i} = \sum_j^{n_{\text{ele}}} \left(-p\, \rho_j^{p-1}\, \frac{\overline{E} - E_{\min}}{\overline{E}}\, \mathbf{u}_j^{\mathsf{T}}\, \overline{\mathbf{K}}_j\, \mathbf{u}_j\, \frac{H(j,\, i)\, v_i}{\sum_k H(j,\, k)\, v_k} \right). \tag{2.11}$$

Since the volume constraint can be written as

$$V(\hat{\boldsymbol{\rho}}) = \sum_i \hat{\rho}_i\, v_i \leq V_f\, \overline{V}, \tag{2.12}$$

the derivative of the volume constraint (2.1c) is obtained analogously with

$$\frac{\partial V(\hat{\boldsymbol{\rho}})}{\partial \rho_i} = \frac{\partial V(\hat{\boldsymbol{\rho}})}{\partial \hat{\rho}_j}\, \frac{\partial \hat{\rho}_j}{\partial \rho_i} = v_i \sum_j^{n_{\text{ele}}} \frac{H(j,\, i)\, v_j}{\sum_k H(j,\, k)\, v_k}\,. \tag{2.13}$$

2.1.6 Optimality Criteria (OC) method

The OC method is formulated on the grounds that if the constraint $\mathbf{0} \leq \hat{\boldsymbol{\rho}} \leq \mathbf{1}$ is inactive, then convergence is achieved when the *Karush–Kuhn–Tucker* (KKT) [80] condition

$$\frac{\partial J(\hat{\boldsymbol{\rho}})}{\rho_i} + \lambda\, \frac{V(\hat{\boldsymbol{\rho}})}{\rho_i} = 0 \tag{2.14}$$

is satisfied for $i = 1, \ldots, n_{\text{ele}}$, where λ is the Lagrange multiplier associated with the volume constraint $V(\hat{\boldsymbol{\rho}})$. This optimality condition can be expressed as $B_i = 1$, and

$$B_i = -\frac{\partial J(\hat{\boldsymbol{\rho}})}{\rho_i} \left(\lambda\, \frac{V(\hat{\boldsymbol{\rho}})}{\rho_i} \right)^{-1}. \tag{2.15}$$

The densities are updated based on the scheme

$$\rho_i^{\text{new}} = \begin{cases} \max(0,\, \rho_i - m) & \text{if } \rho_i\, B_i^{\eta} \leq \max(0,\, \rho_i - m) \\ \min(1,\, \rho_i + m) & \text{if } \rho_i\, B_i^{\eta} \geq \min(1,\, \rho_i - m) \\ \rho_i\, B_i^{\eta} & \text{otherwise}\,, \end{cases} \tag{2.16}$$

where ρ_i^{new} is the updated density in element i, m is a positive move limit, and η is a numerical damping coefficient; $m = 0.2$ and $\eta = 0.5$ are recommended for minimum compliance problem [56]. The Lagrange multiplier λ can be found by solving

$$V(\hat{\boldsymbol{\rho}}) = 0\,, \tag{2.17}$$

where $\hat{\boldsymbol{\rho}} = \hat{\boldsymbol{\rho}}(\boldsymbol{\rho}^{\text{new}}(\lambda))$. With the obtained derivative of the compliance cost function (2.11) and the volume constraint (2.13), the optimised density distribution is determined iteratively with the OC method. The commonly used termination criterion for topology optimisation is

$$\max(|\rho_i^{\text{new}} - \rho_i|) < \epsilon_{\text{top}}, \quad i = 1, 2, \ldots, n_{\text{ele}}\,. \tag{2.18}$$

Normally ϵ_{top} is recommended as 0.01 [77], which ensures that the topology converges sufficiently.

In most problems the convergence of cost function $J(\hat{\boldsymbol{\rho}})$ is somewhat similar to the one shown in Figure 2.6, with the density contour plots at specific steps. From

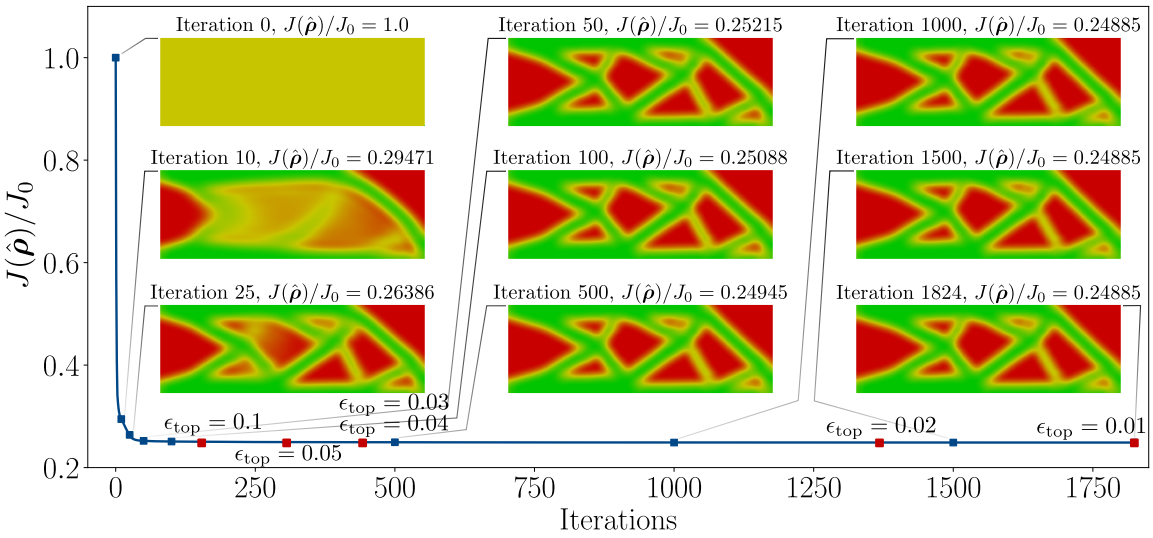


Fig. 2.6 Convergence plot of the cost function for a representative topology optimisation problem. The optimised structure is a cantilever using a mesh with 150×100 elements. The optimisation parameters are chosen as $p = 3$, $R = 6$, $V_f = 0.5$. Red points mark where the optimisation terminate under different torlerances $\epsilon_{\text{top}} = 0.01, 0.02, 0.03, 0.04, 0.05$, and the corresponding termination iterations are $147, 308, 465, 465, 1376, 1824$ respectively. $J_0 = 100.18$ is the cost function value of the initial shape with all elements of $\rho_i = 0.5$.

step 50 until the optimisation terminates at step 1824, the optimised layout remains the same and the cost function value reduces by a negligible amount ($J(\hat{\boldsymbol{\rho}})/J_0$ from 0.25215 to 0.24885, with a decrease of 1.3%). The main reason for such difficulties in convergence (visually as a long flat region in the convergence plot) comes from the SIMP method, as claimed by Andreassen et al. [77]. As far as the density-based approach is concerned, there is yet still no effective solution to such issues [62]. Since the frame optimisation will recover the loss in compliance, less emphasis is addressed on pursuing a highly accurate topology optimised shape. Therefore, there is no need to exhaust the topology optimisation to find the layout with the global minimum cost function value. It is recommended to terminate the topology optimisation at a reasonable number of iterations or to set the tolerance higher in Equation (2.18) when using the proposed approach. For instance, in this thesis, maximum iteration of the OC method solver is 100.

2.2 Review of Frame optimisation

The frame size and layout optimisation can be efficiently performed using gradient-based optimisation techniques, since the derivatives of the cost functions (e.g. structural compliance and maximum stress) and the constraints (e.g. material volume and node positions) are relatively easy to compute. In cases where there is a large number of complex constraints, one can formulate the optimisation problem using *Generalised Geometric Programming* (GPP) [81, 82], *Duffin's condensation formula* [83], or Kreisselmeier-Steinhauser aggregation (KS) function [84, 85], see Appendix B. More options can be found in [86] by Ohsaki, who reviewed a large number of practical truss and frame optimisation methods.

2.2.1 Optimisation formulation

Here spatial frame structures consist of straight beam members connected by joints that can transfer forces and moments. The members can deform by stretching, bending and torsion and are modelled as classical Timoshenko beams, see Appendix A.2. Without loss of generality, combined size and layout optimisation can be posed as an iterative sequential optimisation problem. In each optimisation step, either the size, i.e. cross-section areas, or layout, i.e. joint coordinates, of the members are optimised.

For size optimisation, the optimisation variable can be cross-section area or any other parameters describing the cross-section. The cross-section area is considered

here, which is denoted as

$$\mathbf{A} = (A_1,\, A_2,\, \ldots,\, A_{n_{\text{ele}}}) \in \mathbb{R}^{n_{\text{ele}} \times 1}\,, \tag{2.19}$$

where a component A_i is the cross-section area of member i and the number of members in the mesh is denoted with n_{ele} as in the previous section. In layout optimisation, the optimisation variable is the frame joint coordinate component, which is given as

$$\mathbf{s} = (x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_{n_{\text{nod}}}, y_{n_{\text{nod}}}, z_{n_{\text{nod}}}) \in \mathbb{R}^{3n_{\text{nod}} \times 1}\,, \tag{2.20}$$

where x_i, y_i, z_i form the frame joint coordinate in the global coordinate system of node i, and n_{nod} is the number of nodes in the finite element model. Therefore, (2.20) can be equivalently expressed as

$$\mathbf{s} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{\text{nod}}) \in \mathbb{R}^{3n_{\text{nod}} \times 1}\,, \tag{2.21}$$

where $\boldsymbol{x}_i = (x_i,\, y_i,\, z_i)$ is the coordinate of node i.

The size optimisation problem has the same structure as the topology optimisation problem (2.1), namely,

$$\min_{\mathbf{A}_l \le \mathbf{A} \le \mathbf{A}_u} \quad J(\mathbf{A},\, \mathbf{s}) = \mathbf{f}^{\mathsf{T}}\, \mathbf{u} = \mathbf{u}^{\mathsf{T}}\, \mathbf{K}\, \mathbf{u} \tag{2.22a}$$

$$\text{subject to} \quad \mathbf{K}\, \mathbf{u} = \mathbf{f} \tag{2.22b}$$

$$\frac{V(\mathbf{A},\, \mathbf{s})}{\overline{V}} \le V_f\,. \tag{2.22c}$$

The two bounds, $\mathbf{A}_l$ and $\mathbf{A}_u$, for cross-section areas constrain the member sizes to fit the manufacturing requirements. Similarly, the layout optimisation with the optimisation variable $\mathbf{s}$ reads

$$\min_{\mathbf{s}_l \le \mathbf{s} \le \mathbf{s}_u} \quad J(\mathbf{A},\, \mathbf{s}) = \mathbf{f}^{\mathsf{T}}\, \mathbf{u} = \mathbf{u}^{\mathsf{T}}\, \mathbf{K}\, \mathbf{u} \tag{2.23a}$$

$$\text{subject to} \quad \mathbf{K}\, \mathbf{u} = \mathbf{f} \tag{2.23b}$$

$$\frac{V(\mathbf{A},\, \mathbf{s})}{\overline{V}} \le V_f\,. \tag{2.23c}$$

The lower and upper bounds for coordinate components, $\mathbf{s}_l$ and $\mathbf{s}_u$, are used for keeping the nodes inside a prescribed domain.

The frame consists of n_{ele} beam elements with the element stiffness matrix $\mathbf{K}_i \in \mathbb{R}^{12\times12}$ and vector $\mathbf{f}_i \in \mathbb{R}^{12\times1}$. Each element stiffness matrix $\mathbf{K}_i$ is obtained from a local stiffness matrix $\mathbf{K}_i^l \in \mathbb{R}^{12\times12}$ formulated in a coordinate system attached to the element with the index i. The local matrix $\mathbf{K}_i^l$ with the displacements and rotations of the two end nodes of the beam element as degrees of freedom is given in A.2. In the local x_i^l, y_i^l, and z_i^l coordinate system the beam axis is assumed to be aligned with the x_i^l axis. The matrix $\mathbf{K}_i^l$ is transformed to the global x, y and z coordinate system according to

$$\mathbf{K}_i = \mathbf{\Lambda}_i \, \mathbf{K}_i^l \, \mathbf{\Lambda}_i^{\mathsf{T}} \tag{2.24}$$

with the block-diagonal transformation matrix

$$\mathbf{\Lambda}_i = \operatorname{diag}(\boldsymbol{\lambda}_i, \, \boldsymbol{\lambda}_i, \, \boldsymbol{\lambda}_i, \, \boldsymbol{\lambda}_i) \in \mathbb{R}^{12\times12} \, . \tag{2.25}$$

The rotation matrix $\boldsymbol{\lambda}_i \in \mathbb{R}^{3\times3}$ can be chosen with

$$\boldsymbol{\lambda}_i = \begin{pmatrix} \cos\alpha_i \cos\beta_i & -\sin\alpha_i & \cos\alpha_i \sin\beta_i \\ \sin\alpha_i \cos\beta_i & \cos\alpha_i & \sin\alpha_i \sin\beta_i \\ -\sin\beta_i & 0 & \cos\beta_i \end{pmatrix} , \tag{2.26}$$

which is composed of the two elemental rotations α_i around the z_i^l axis and β_i around y_i^l axis. The details of these two spatial angles can be found in Appendix A.2. Note that the element stiffness matrix $\mathbf{K}_i$ is a function of both cross-section areas and coordinate components, while the transformation matrix $\mathbf{\Lambda}_i$ is a function of only coordinate components.

2.2.2 Sensitivity analysis

In size optimisation, similar as in topology optimisation, the derivative, or sensitivity, of the compliance cost function (2.22a) is needed, which reads

$$\frac{\partial J(\mathbf{A}, \, \mathbf{s})}{\partial A_i} = -\mathbf{u}^{\mathsf{T}} \frac{\partial \mathbf{K}(\mathbf{A}, \, \mathbf{s})}{\partial A_i} \mathbf{u} = -\sum_{j=1}^{n_{\text{ele}}} \mathbf{u}_j^{\mathsf{T}} \frac{\partial \mathbf{K}_j(\mathbf{A}, \, \mathbf{s})}{\partial A_i} \mathbf{u}_j \, . \tag{2.27}$$

Substituting (2.24) into (2.27) gives

$$\frac{\partial J(\mathbf{A}, \, \mathbf{s})}{\partial A_i} = -\sum_{j=1}^{n_{\text{ele}}} \mathbf{u}_j^{\mathsf{T}} \left(\mathbf{\Lambda}_j \frac{\partial \mathbf{K}_j^l(\mathbf{A}, \mathbf{s})}{\partial A_i} \mathbf{\Lambda}_j^{\mathsf{T}} \right) \mathbf{u}_j \, , \tag{2.28}$$

where $i, j = 1, 2, \ldots, n_{\text{ele}}$ are element indices and $\mathbf{u}_j$ is the vector of nodal displacements and rotation of element j. Note that $\partial \mathbf{K}_j^l(\mathbf{A}, \mathbf{s})/\partial A_i$ vanishes when $j \neq i$.

The sensitivity of the cost function in layout optimisation is derived with respect to a nodal coordinate component as

$$\begin{aligned}\frac{\partial J(\mathbf{A},\,\mathbf{s})}{\partial s_i} &= -\mathbf{u}^{\mathsf{T}} \frac{\partial \mathbf{K}(\mathbf{A},\,\mathbf{s})}{\partial s_i}\,\mathbf{u} = -\sum_{j=1}^{n_{\text{ele}}} \mathbf{u}_j^{\mathsf{T}} \frac{\partial \mathbf{K}_j(\mathbf{A},\,\mathbf{s})}{\partial s_i}\,\mathbf{u}_j \\ &= -\sum_{j=1}^{n_{\text{ele}}} \mathbf{u}_j^{\mathsf{T}} \left(\frac{\partial \mathbf{\Lambda}_j}{\partial s_i}\,\mathbf{K}_j^l\,\mathbf{\Lambda}_j^{\mathsf{T}} + \mathbf{\Lambda}_j \frac{\partial \mathbf{K}_j^l}{\partial s_i}\,\mathbf{\Lambda}_j^{\mathsf{T}} + \mathbf{\Lambda}_j\,\mathbf{K}_j^l \frac{\partial \mathbf{\Lambda}_j^{\mathsf{T}}}{\partial s_i} \right) \mathbf{u}_j\,,\end{aligned} \tag{2.29}$$

where $i = 1, 2, \ldots, 3n_{\text{nod}}$. Note that $\partial \mathbf{\Lambda}_j/\partial s_i$ and $\partial \mathbf{K}_j^l/\partial s_i$ vanish when s_i is not the coordinate component of the node on element j.

The volume constraint can be interpreted as

$$V(\mathbf{A},\,\mathbf{s}) = \sum_{i}^{n_{\text{ele}}} A_i L_i \leq V_f \overline{V}, \tag{2.30}$$

where L_i is the length of the member i. Therefore, in size optimisation, the derivative of the volume constraint with respect to a cross-section area is

$$\frac{\partial V(\mathbf{A},\,\mathbf{s})}{\partial A_i} = L_i\,, \tag{2.31}$$

where $i = 1, 2, \ldots, n_{\text{ele}}$ is the element index.

In layout optimisation the derivative of volume constraint with respect to a nodal coordinate component is

$$\frac{\partial V(\mathbf{A},\,\mathbf{s})}{\partial s_i} = \sum_{j=1}^{n_{\text{ele}}} A_j \frac{\partial L_j(\mathbf{s})}{\partial s_i}\,, \tag{2.32}$$

where $i = 1, 2, \ldots, 3n_{\text{nod}}$. $\partial L_j(\mathbf{s})/\partial s_i$ vanishes when s_i is not the coordinate component of the node on element j. With Equations (A.8), (A.9), (A.6) and (A.7), the derivatives above can be determined in a straightforward way.

2.2.3 Sequential size and layout optimisation

In order to obtain a frame structure with both member cross-section sizes and layouts optimised, sequential size and layout optimisation is used. The frame optimisation starts with size optimisation, and once the current optimisation converges, the optimisation proceeds to the next *step*, layout optimisation; then the optimisation takes another

size optimisation after the layout optimisation terminates and this process is repeated until the sequential optimisation converges. Iterations inside individual size and layout optimisation are called as *sub-steps* to distinguish from the aforementioned *step*, and the structural compliance of one step is referred to as the minimised compliance in such step. The termination criterion for sub-step/step i is given as

$$0 \leq \frac{J_{i-1} - J_i}{J_i} < \epsilon_{\text{frame}}, \tag{2.33}$$

where J_i is the structural compliance at the i sub-step/step, and ϵ_{frame} is the user-defined tolerance. In this thesis, $\epsilon_{\text{frame}} = 10^{-4}$ is used. As for the optimisation solver, one of the gradient-based methods can be chosen, such as the SQP or MMA, see Appendices C.1 and C.2.

During the layout optimisation, the zero-stress (overall stress $< 10^{-6}$) members are removed. Besides, nodes may move close to other nodes, which leads to short members. A member shorter than a fraction ζ of the total length of all members sharing the same node is considered as a short member, and its two end nodes are merged. This user-defined ratio ζ is named as *merge ratio*, and in this thesis $\zeta = 1/20$ is used. The volume of the removed short member is uniformly redistributed and added to its adjacent members.

2.2.4 Geometric constraints

The loading and boundary conditions in topology optimisation are equivalently applied to frame optimisation. In most cases, besides the volume constraint (2.23c), there are also geometric constraints. In topology optimisation, the elements with Young's modulus of E_{min} are used to represent the regions outside design domains. In frame layout optimisation, if the design domain has no straightforward shape, using only the upper/lower bounds for nodal coordinates as geometric constraints is insufficient.

a. Level-set function

Here the geometric constraints are described using *level-set functions*. Level-set function is an implicit geometric representation. Complex shapes, which cannot be easily described in an explicit form, can be conveniently described using an implicit geometric representation. Frequently, the level-set function is defined on a grid and it returns for a point on the grid its distance to the nearest point on the boundary; the sign of the level-set function value indicates whether the point is inside (positive), outside the

shape (negative) or right on the boundary surface (zero). The level-set function $L_\Omega(\boldsymbol{x})$ for a given domain Ω has the following properties

$$L_\Omega(\boldsymbol{x}) = \begin{cases} \min_{\boldsymbol{y}\in\partial\Omega}(\|\boldsymbol{x}-\boldsymbol{y}\|) & \text{if } \boldsymbol{x}\in\Omega \\ 0 & \text{if } \boldsymbol{x}\in\partial\Omega \\ -\min_{\boldsymbol{y}\in\partial\Omega}(\|\boldsymbol{x}-\boldsymbol{y}\|) & \text{otherwise}\,, \end{cases} \tag{2.34}$$

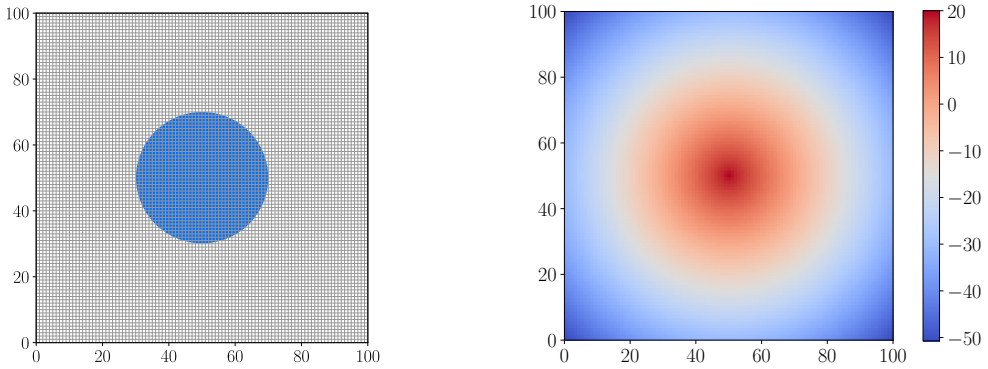
where $\boldsymbol{x}$ and $\boldsymbol{y}$ ($\boldsymbol{y}$ is on the boundary $\partial\Omega$) are coordinate vectors, and $\|\boldsymbol{x}-\boldsymbol{y}\|$ gives the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. In this thesis, the value of a level-set function gives the Euclidean distance, but it can also be an user-defined distance-like value.

For instance, the level-set function for a planar circular domain

$$\Omega_{\text{circle}}(\boldsymbol{x}) = \{\boldsymbol{x}=(x,y,z)\,|\,(x-50)^2+(y-50)^2-20^2\leq 0,\ z=0\}\,,$$

on a 2D grid of 100×100 cells is shown using a contour plot in Figure 2.7b, and the corresponding grid is shown in Figure 2.7a. Any point $\boldsymbol{x}$ inside the circular domain has a non-negative level-set function value, i.e. $L_{\Omega_{\text{circle}}}(\boldsymbol{x})\geq 0$, and $L_{\Omega_{\text{circle}}}(\boldsymbol{x})<0$ for any point $\boldsymbol{x}$ outside the domain.

In some cases, a level-set function may have *singular points*, where the gradient of the level-set function is not well-defined or is infinite. Singular points can be found



(a) Domain defined on a grid (b) Level-set function value

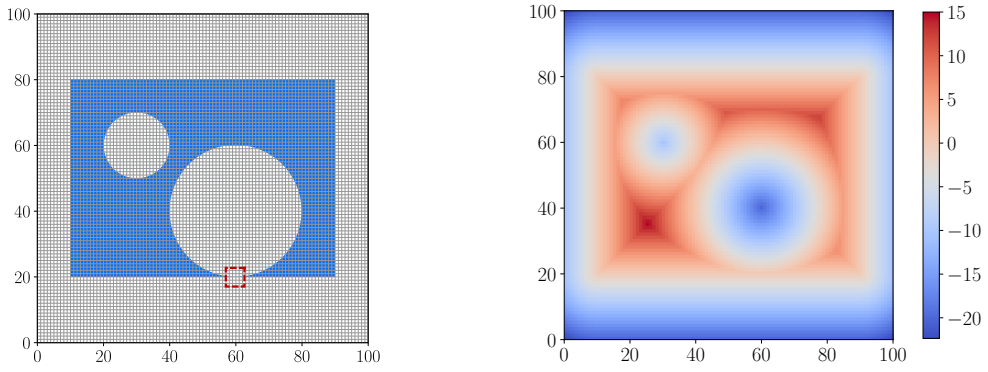
Fig. 2.7 Level-set representation of a circular domain. (a) shows the object of a blue circle with radius of 20 centred at $(50, 50)$ on a 2D grid with 100×100 cells. (b) gives the contour plot of the level-set function (2.34) for the circular domain. The maximum level-set function value of 20 (its magnitude equals the radius of the circle) is at the centre of the circle, while points at four grid corners have the minima of -50.711 (the distance between the corner points and the circle centre).

at the tangent points/lines between domain edges [87]. For example, the domain, as shown in Figure 2.8 is a rectangle subtracting two circles. This domain is defined as

$$\begin{aligned}\Omega_0(\boldsymbol{x}) = \{\boldsymbol{x} = (x, y, z) \,|\, & 10 \leq x \leq 90,\ 20 \leq y \leq 80,\ z = 0, \\ & (x-30)^2 + (y-60)^2 - 10^2 \geq 0, \\ & (x-60)^2 + (y-40)^2 - 20^2 \geq 0\}\,.\end{aligned}$$

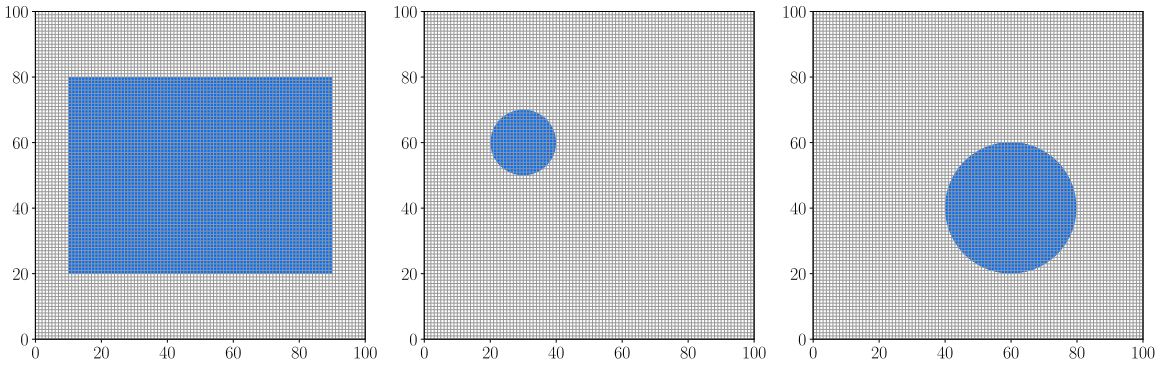
In Figure 2.7b, the singular point in the level-set function L_{Ω_0} is highlighted in the red window. Such singular point lies where the circle edge is tangent to the rectangle edge. Most structural optimisation methods are gradient-based, which require valid gradients of constraint functions. When using a level-set function to formulate the geometric constraints, singular points with infinite gradients may lead to numerical instabilities. In this case the shapes can be considered separately, e.g. a point inside domain Ω_0 can be also described as this point is inside the rectangle as shown in Figure 2.9a and outside two circles as shown in Figures 2.9b and 2.9c.

Note that domains that are described with a parametric polygon mesh can be converted into an level-set function using *scan conversion* method [88]. Scan conversion generates the signs of grid vertices by first intersecting the parametric geometry with planes that coincide with each grid point along a coordinate direction. The resulting intersection curve is used to decide on the sign of the grid points in the plane of the curve. Then the acquired signs are assigned to the minimum distance of grid vertices



(a) Domain defined on a grid (b) Level-set function value

Fig. 2.8 Level-set representation of the domain Ω_0. The 2D grid consists of 100×100 cells. The domain is highlighted in blue in (a), and the position of the singular point of the level-set function is indicated with the red dashed box in (b). At the singular point the gradient is infinite. Corresponding level-set function values are displayed in (b).



(a) (b) (c)

Fig. 2.9 Three simple shapes used for defining the domain Ω_0. The 2D grid consists of 100×100 cells. The three domains are defined as (a): $\Omega_{\mathrm{a}}(\boldsymbol{x}) = \{\boldsymbol{x} = (x, y, z) \,|\, 10 \leq x \leq 90,\ 20 \leq y \leq 80,\ z = 0\}$, (b): $\Omega_{\mathrm{b}}(\boldsymbol{x}) = \{\boldsymbol{x} = (x, y, z) \,|\, (x-30)^2 + (y-60)^2 - 10 \leq 0,\ z = 0\}$ and (c): $\Omega_{\mathrm{c}}(\boldsymbol{x}) = \{\boldsymbol{x} = (x, y, z) \,|\, (x-60)^2 + (y-40)^2 - 20 \leq 0,\ z = 0\}$.

to the ones on the boundary surfaces; this gives the level-set function on the given domain.

b. Geometric constraint formulation

For limiting a frame joint i within a design domain Ω during the optimisation, the constraint function can be formulated as

$$g^{\mathrm{nod}}(\boldsymbol{x}_i) = -L_\Omega(\boldsymbol{x}_i) \leq 0\,, \tag{2.35}$$

where $L_\Omega(\cdot)$ is the level-set function for the design domain Ω, $\boldsymbol{x}_i$ is the coordinate of the node $i = 1, 2, \ldots, n_{\mathrm{nod}}$ and n_{nod} is the total number of frame joints. For the sake of clarity, uppercase subscripts are used for element indices, whereas lowercase subscripts are used for nodal indices.

However, nodal position constraints (2.35) cannot ensure that the frame members protrude the design domain. It is quite common that two nodes are inside the domain, while part of the beam connecting these two nodes is outside the domain, see Figure 2.10a. The volumes of beams need to be considered when applying constraints. As shown in Figure 2.10b, the medial axis of the beam is outside the hole, but part of the beam volume is inside the hole. As a result, for a straight frame member I, consisting of two joints i and j, the geometric constraint is described as

$$g^{\mathrm{ele}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = -L_\Omega((1-t)\,\boldsymbol{x}_i + t\,\boldsymbol{x}_j) + r_I \leq 0\,, \tag{2.36}$$

(a) (b)

Fig. 2.10 Two cases of a frame member protruding the design domain. The design domain is coloured in blue, and the beam axis is defined by the segment $(1-t)\boldsymbol{x}_i + t\boldsymbol{x}_j$ with $t \in (0,1)$. (a) shows even when the beam nodes are in the domain, the centreline of the beam penetrates the hole. The red transparent region in (b) with the offset r_I presents the volume of the beam, part of which protrudes the domain.

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are coordinate of joint i and j respectively, $t \in (0,1)$ is a scalar for defining the points on the beam, and r_I is an offset assigning a thickness to the element I. For example, for beams using circular cross-sections, as assumed in this thesis, the offset in (2.36) can be the radius of the circular cross-section

$$r_I = \sqrt{\frac{A_I}{\pi}}, \tag{2.37}$$

where A_I is the cross-section area of element I. In practice, one beam member is divided into 20 segments to check the level-set function values at the middle 19 points along the frame member. In this case, in total $n_{\text{nod}} + 19n_{\text{ele}}$ constraints are considered. When the number of geometric constraint functions become very large, it may induce numerical difficulties to the optimisation process [85]. This problem can be overcome by using constraint aggregation methods, e.g. Kreisselmeier-Steinhauser (KS) aggregation function, see Appendix B. Note that if along a beam the sign of the level-set function gradient component changes several times, it is recommended to increase the number of segments.

c. Sensitivity analysis of implicit geometric constraints

In order to embed the geometric constraints (2.35) into the optimisation formulation (2.22) and (2.23), gradient of the constraint functions are required. The first order derivative of the geometric constraint at the node i with respect to a cross-section area is

$$\frac{\partial g^{\text{nod}}(\boldsymbol{x}_i)}{\partial A_J} = -\frac{\partial L_\Omega(\boldsymbol{x}_i)}{\partial A_J} = 0\,, \tag{2.38}$$

and the derivative with respect to a nodal coordinate is

$$\frac{\partial g^{\mathrm{nod}}(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_k} = \begin{cases} -\nabla L_\Omega(\boldsymbol{x}_i) & \text{if } k = i \\ 0 & \text{otherwise}\,. \end{cases} \tag{2.39}$$

The first order derivative of the geometric constraint at the element I consisting of two nodes i with coordinate $\boldsymbol{x}_i$ and j with coordinate $\boldsymbol{x}_j$, with respect to a cross-section area is

$$\frac{\partial g^{\mathrm{ele}}(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial A_J} = \begin{cases} \frac{1}{2}\sqrt{\frac{1}{\pi A_I}} & \text{if } J = I \\ 0 & \text{otherwise}\,; \end{cases} \tag{2.40}$$

and the derivative with respect to a nodal coordinate using the chain rule gives

$$\frac{\partial g_I^{\mathrm{ele}}(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \boldsymbol{x}_k} = \begin{cases} -(1-t)\,\nabla L_\Omega((1-t)\boldsymbol{x}_i + t\boldsymbol{x}_j) & \text{if } k = i \\ -t\,\nabla L_\Omega((1-t)\boldsymbol{x}_i + t\boldsymbol{x}_j) & \text{if } k = j \\ \mathbf{0} & \text{otherwise}\,. \end{cases} \tag{2.41}$$

2.2.5 Frame optimisation examples

Three examples are provided to investigate the convergence and robustness of the proposed optimisation process. Table 2.1 gives the material properties used in these examples.

Table 2.1 Material properties for frame optimisation

Young's modulus E	Poisson ratio ν
2.1×10^5	0.3

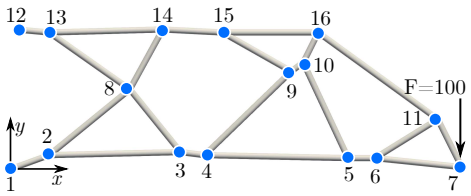
a. Two-dimensional cantilever

The first example is a cantilever with frame nodes poorly positioned. The geometry and coordinates of the beam nodes can be seen in Table 2.2. The cantilever has a vertical force at the right bottom corner of 100, and the left side is fixed.

Here, only the first two steps (one size and one layout optimisation) of the frame optimisation process are presented to study particularly the effects of the optimisation on the member. The frame optimisation is expected to redistribute the stress more

evenly and make the bending effects less dominant. Without any prior knowledge of the cross-section sizes, all frame members are initialised with a uniform cross-section area of 15.545 giving the target volume of 9000. It is assumed that circular cross-sections for all members for simplicity, but one can easily extend the presented idea into arbitrary cross-sections. The material properties and optimisation parameters are listed in Tables 2.1 and 2.3.

Table 2.2 Node coordinates of the initial 2D cantilever



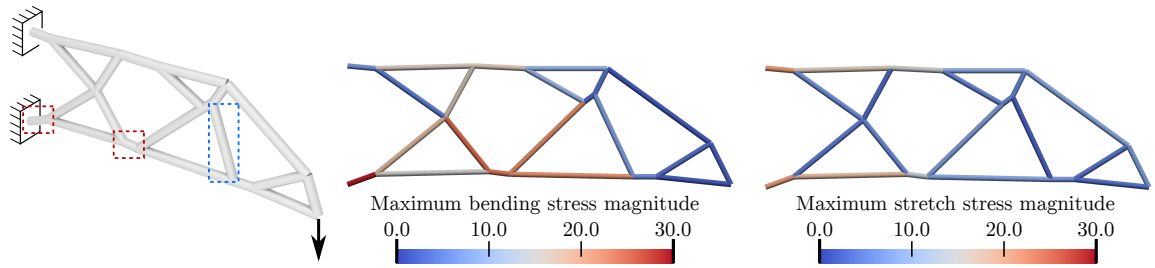
Node	x	y	Node	x	y
1	0.0	0.0	9	92.5	32.5
2	11.0	4.5	10	96.5	35.5
3	55.5	5.5	11	141.0	16.5
4	63.0	4.5	12	0.0	46.5
5	111.5	3.5	13	11.5	45.5
6	120.5	3.5	14	49.5	46.5
7	150.0	0.0	15	69.0	45.5
8	38.5	26.5	16	101.0	45.5

Table 2.3 Optimisation parameters for 2D frame optimisations in Section 2.2.5

Minimum area $A_{\min}$	Maximum area $A_{\max}$	Merge ratio ζ	Tolerance ϵ_{frame}	Volume constraint
$1.5^2\pi$	–	1/20	10^{-4}	$V \leq 9000$

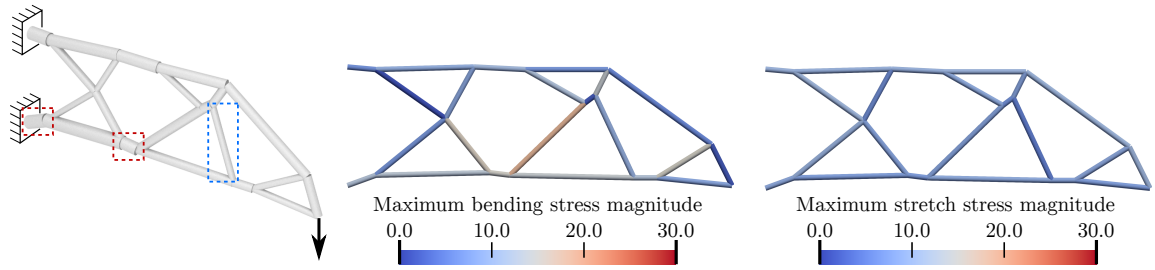
As cen be seen in Figure 2.11c and 2.11b, due to the flawed design of the cantilever, the member stresses in the initial frame are relatively high. Both the maximum bending stress of 29.314 and maximum axial stress of 21.479 occur in the short beam on the left bottom corner as highlighted in the left red window in Figure 2.11a. The stress values used here are referred as the absolute magnitudes of maximum stresses along beams. Note that the member bending stresses have a similar magnitude like the axial stresses, and there is a large spread in the stress magnitudes, i.e. bending stress is 11.890(mean)±8.786(variance) and axial stress is 8.960 ± 5.819, see Figures 2.11b and 2.11c.

Next, the material distribution in the frame structure is optimised using size optimisation. This helps to adjust member sizes according to the member stresses. For instance, the cross-sections of members in the red windows in Figure 2.11a, which have relatively high stress levels, are enlarged as shown in the red windows in Figure 2.12a. In contrast, due to the volume constraint, the members bearing relatively low stresses



(a) Frame model (b) Bending stress (c) Axial stress

Fig. 2.11 Geometry and stresses of the initial cantilever frame. The structural compliance is 5.5217. Model (a) shows the actual cross-sectional sizes. (b) Bending stress has mean 11.890, variance 8.786 and maximum 29.314. (c) Axial stress has mean 8.9654, variance 5.8139 and maximum 21.479.



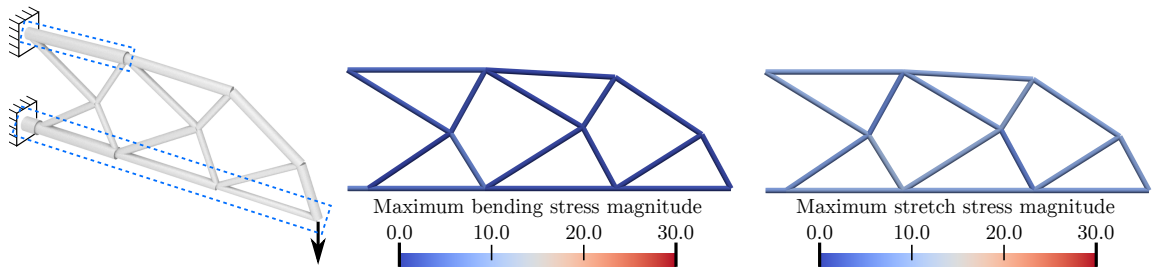
(a) Frame model (b) Bending stress (c) Axial stress

Fig. 2.12 Geometry and stresses of the size optimised cantilever frame. The structural compliance reduces to 3.9746. Model (a) shows the actual cross-sectional sizes. (b) Bending stress has mean 9.6691, variance 4.8068 and maximum 18.7272. (c) Axial stress has mean 8.3985, variance 1.4511 and maximum 10.169.

become thinner, e.g. the one in the blue windows in Figures 2.11a and 2.12a. The size optimised frame has the stresses depicted in Figures 2.12b and 2.12c with a stress distribution that is more even (bending stress: 9.6691 ± 4.8068, axial stress: 8.3985 ± 1.4511) compared to Figures 2.11b and 2.11c.

After the size optimisation converges, the member cross-sectional areas are kept and the layout optimisation starts. Size optimisation can efficiently optimise the axial stress among members, but it has its limitation in optimising bending stresses, since the magnitudes of bending moments are closely related to the frame layout. As is visually evident in Figure 2.13a, the layout optimisation especially makes the top and bottom beams horizontally aligned, as shown in the blue window, which significantly reduces the bending moments in these beams. This can be further validated in Table 2.4, i.e. node 1, 2, 3, 4 and 5 lie on a flat line, and node 9 and 10 lie on another flat line.

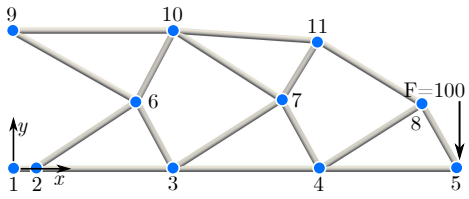
Figure 2.13b demonstrates a dramatic drop in the bending stresses, with the bending stress distribution reducing to 1.6508 ± 1.6275. As can be seen in Figure 2.13c the layout optimisation affects the axial stresses less. After the layout optimisation, the frame becomes stretch dominant.



(a) Frame model (b) Bending stress (c) Axial stress

Fig. 2.13 Geometry and stresses of the layout optimised cantilever frame. The structural compliance is 2.9575. Model (a) shows the actual cross-sectional sizes. (b) Bending stress has mean 1.6508, variance 1.6275 and maximum 5.4598. (c) Axial stress has mean 7.9405, variance 1.2609 and maximum 10.049.

Table 2.4 Node coordinates of the optimal 2D cantilever



Node	x	y	Node	x	y
1	0.0	0.0	7	90.54	23.55
2	8.24	0.0	8	137.32	22.02
3	54.41	0.0	9	0.0	46.5
4	103.23	0.0	10	54.74	46.5
5	150.0	0.0	11	102.64	43.17
6	42.29	22.96			

Altering the merge ratio ζ for short beams lead to different layout-optimised shape. Normally the choice of ζ is in the user's hand, and it directly reflects the user's acceptance level of short beams. Note that a too low or too high merge ratio may adversely change the topology of the frame, and based on the examples in this thesis, $\zeta = 1/20$ is recommended.

b. Two-dimensional simply-supported frame

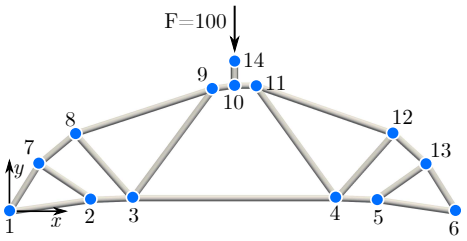
In the previous example, first two steps are used to demonstrate that axial stresses are reduced mainly by size optimisation and bending stresses mainly by layout optimisation.

A 2D simply-supported frame is used here to demonstrate a complete sequential optimisation approach.

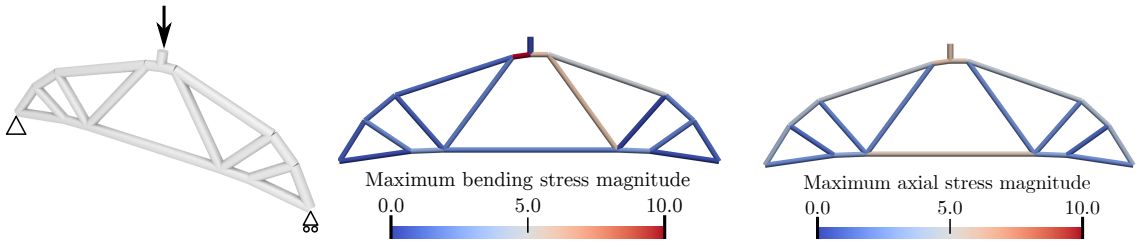
The initial layout of the frame is shown in Table 2.5, which is intentionally chosen not to be strictly symmetric, e.g. see node 9 and 11. A vertical force of 100 is applied at the mid-span. Material properties and optimisation parameters are identical to the ones in Tables 2.1 and 2.3.

The optimisation starts with the initial frame with uniform cross-section areas of 17.064 to give the volume of 9000, see Figure 2.14a. After 41 iterations of size optimisation (S), 136 iterations of layout optimisation (L), 27 iterations of S, 108 iterations of L and 29 iterations of S, the optimisation gives the optimal frame structure shown in Figure 2.15. During the optimisation, as expected, the stress distribution is optimised and symmetry of the frame is recovered. Comparing Figures 2.15b and 2.15c with Figures 2.14b and 2.14c, it is evident that the optimisation reduces the variations of both bending stresses and axial stresses. Moreover, the stress distribution in the final frame is stretch dominant, with axial stresses of 2.7841 ± 0.075846 and bending stresses of 0.51343 ± 0.25604. In terms of the frame layout, the node coordinates of

Table 2.5 Node coordinates of the initial 2D simply-supported frame

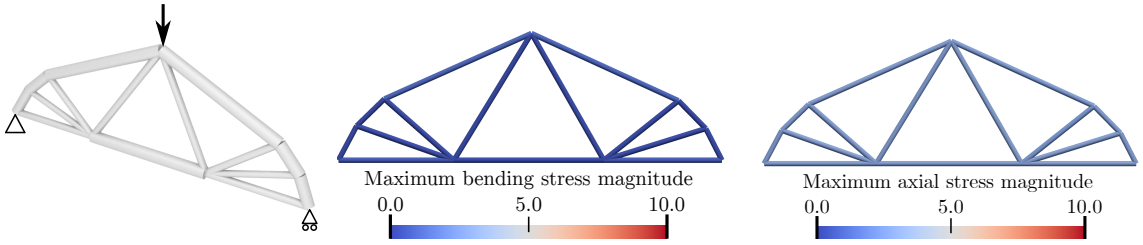


Node	x	y	Node	x	y
1	0.00	0.00	8	22.00	26.00
2	28.00	4.00	9	68.00	41.00
3	41.00	4.50	10	75.00	42.00
4	109.00	4.50	11	82.50	42.50
5	122.50	4.00	12	129.0	26.00
6	150.00	0.00	13	139.50	16.50
7	10.00	16.00	14	75.0	49.50



(a) Frame model (b) Bending stress (c) Axial stress

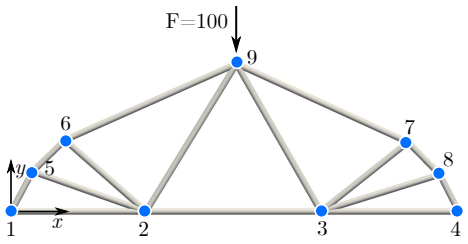
Fig. 2.14 Geometry and stresses of the initial simply-supported frame. Model (a) shows the actual cross-sectional sizes. (b) Bending stress has mean 2.3301, variance 2.5907 and maximum 10.545. (c) Axial stress has mean 3.3287, variance 1.5689 and maximum 5.8602.



(a) Frame model (b) Bending stress (c) Axial stress

Fig. 2.15 Geometry and stresses of the final optimised simply-supported frame. Model (a) shows the actual cross-sectional sizes. (b) Bending stress has mean 0.51343, variance 0.25604 and maximum 1.0627. (c) Axial stress has mean 2.7841, variance 0.075846 and maximum 2.9219.

Table 2.6 Node coordinates of the optimal 2D simply-supported frame



Node	x	y	Node	x	y
1	0.00	0.00	6	18.38	24.19
2	45.38	0.00	7	132.61	23.49
3	103.62	0.00	8	143.20	12.74
4	150.00	0.00	9	75.0	49.50
5	7.60	13.77			

the optimal frame listed in Table 2.6 confirm the symmetry of the frame, e.g. node 2, 5 and 6 are symmetric to node 3, 8 and 7 about the line $x = 75.00$, respectively. Note that, the layout optimisation also adjusts all bottom members to lie on a flat line, i.e. node 1, 2, 3 and 4 are now on the line $y = 0.0$. This establishes an efficient load path connecting the supports at the node 1 and 4, and largely increases the horizontal stiffness of the frame.

Figure 2.16 shows the change in compliance during the optimisation along with the intermediate layouts and corresponding member cross-sectional areas. The compliance continues reducing until it converges at the value of 0.33399; the compliance dropping from 0.63310 to 0.33399 achieves a reduction of 47.25%. The small jump in the convergence curve at iteration 177 (termination sub-step in the second step) is due to the short edge merges. The merging changes the connectivity of the frame, and the cross-sectional areas become sub-optimal after the merging. But the subsequent size optimisation (the third step) recovers the optimality. The entire optimisation terminates when the compliance in fifth step is sufficiently close to the compliance in fourth step, i.e. criterion (2.33) is satisfied.

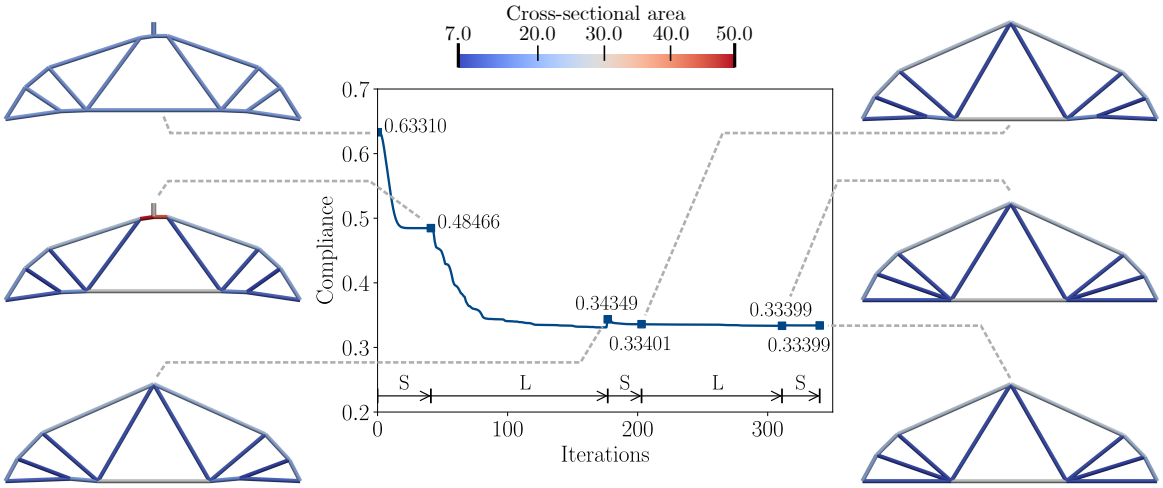


Fig. 2.16 Convergence of the compliance during the sequential size (S) and layout (L) optimisation of the simply-supported frame. The optimisation contains five sub-steps: 41(S), 136(L), 27(S), 108(L) and 29(S). The layouts and cross-sectional areas of intermediate structures during the optimisation are shown on the sides.

c. Three-dimensional frame with implicit geometric constraints

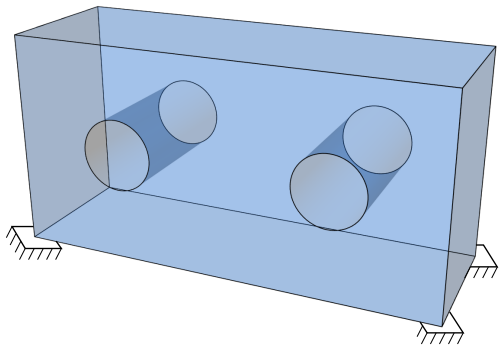
Besides the 2D examples, a 3D frame is considered here with its four corners fixed and an applied downward force of 1000 at its mid-span, as shown in Table 2.8. Again, the frame is not strictly symmetric, e.g. either node 2 and 6 are not symmetric about the line $y = 20.0$, or node 9 and 11 are not symmetric about the line $x = 60.0$. The material properties are shown in Table 2.1 and optimisation parameters are in Table 2.7.

Table 2.7 Optimisation parameters for 3D frame optimisation

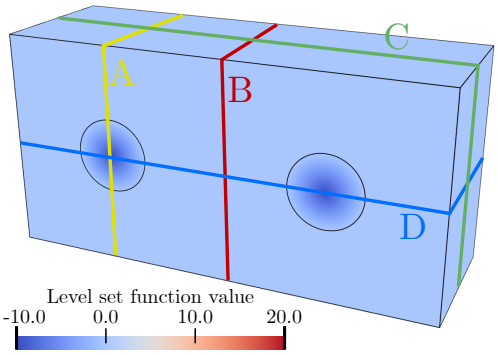
Minimum area $A_{\min}$	Maximum area $A_{\max}$	Merge ratio ζ	Tolerance ϵ_{frame}	Volume constraint
$2^2\pi$	$4^2\pi$	$1/20$	10^{-4}	$V \leq 20000$

The 3D frame is constrained to lie within the design domain given in Figure 2.17a, which consists of a cuboid of size $120 \times 40 \times 60$ and two cylindrical holes of radius of 10 centred at $(30, 20, 30)$ and $(90, 20, 30)$. A level-set function is used to enforce the geometric constraints in this example. The level-set function for the design domain is computed as shown in Figure 2.17b using openVDB [89]. During optimisation the level-set function evaluated along each of the members of the frame is required to be non-negative, i.e. conditions (2.35) and (2.36) must be satisfied.

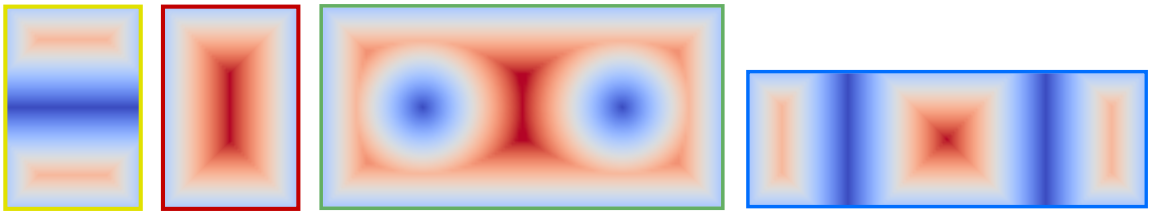
The initial frame layout is given in Table 2.8, with each member having a uniform cross-section area of 25.757 giving the volume of 20000. All members initially lie within the given design domain. The two cylindrical holes are highlighted using transparent blue cylinders in Figures 2.18a and 2.19a. Due to the asymmetry of the frame structure, there are torsional stresses with a distribution of 4.1508 ± 2.3121, and the bending and axial stresses are not evenly distributed. Especially the member on the back-left bottom (shown in the blue windows in Figures 2.18c and 2.18d) bears the maximum



(a) Design domain



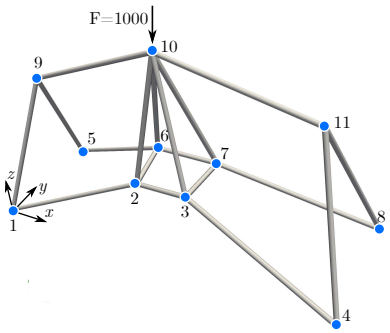
(b) Level-set function values on the domain



(c) Slice A (d) Slice B (e) Slice C (f) Slice D

Fig. 2.17 Design domain and the isocontours of its level set function. (c), (d), (e) and (f) show the level-set function values on the indicated slices.

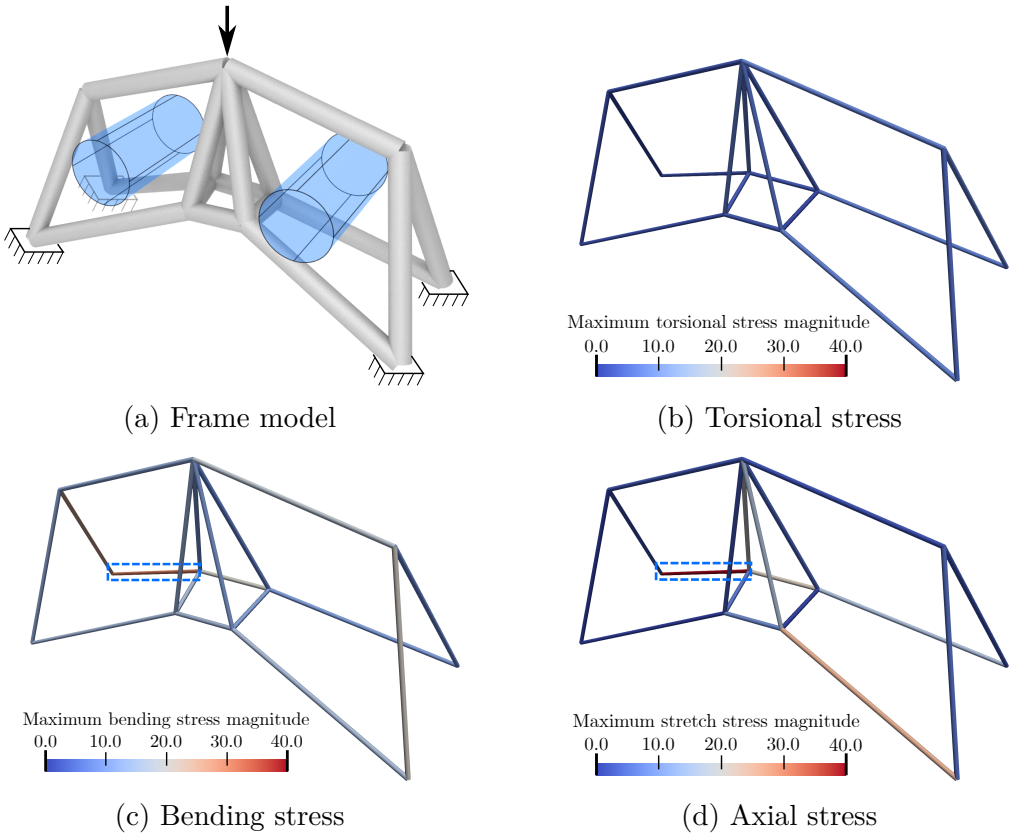
Table 2.8 Node coordinates of the initial 3D frame



Node	x	y	z	Node	x	y	z
1	0.00	0.00	0.00	7	69.93	30.65	18.28
2	51.99	11.17	19.11	8	120.00	40.00	0.00
3	71.72	8.67	21.80	9	11.51	19.82	44.82
4	120.00	0.00	0.00	10	60.00	20.00	60.00
5	0.00	40.00	0.00	11	108.01	20.05	46.64
6	48.56	28.91	20.28				

bending stress of 22.121 as well as the maximum axial stress of 39.5428, both of which are far from the mean of 11.243 and 9.7061 respectively.

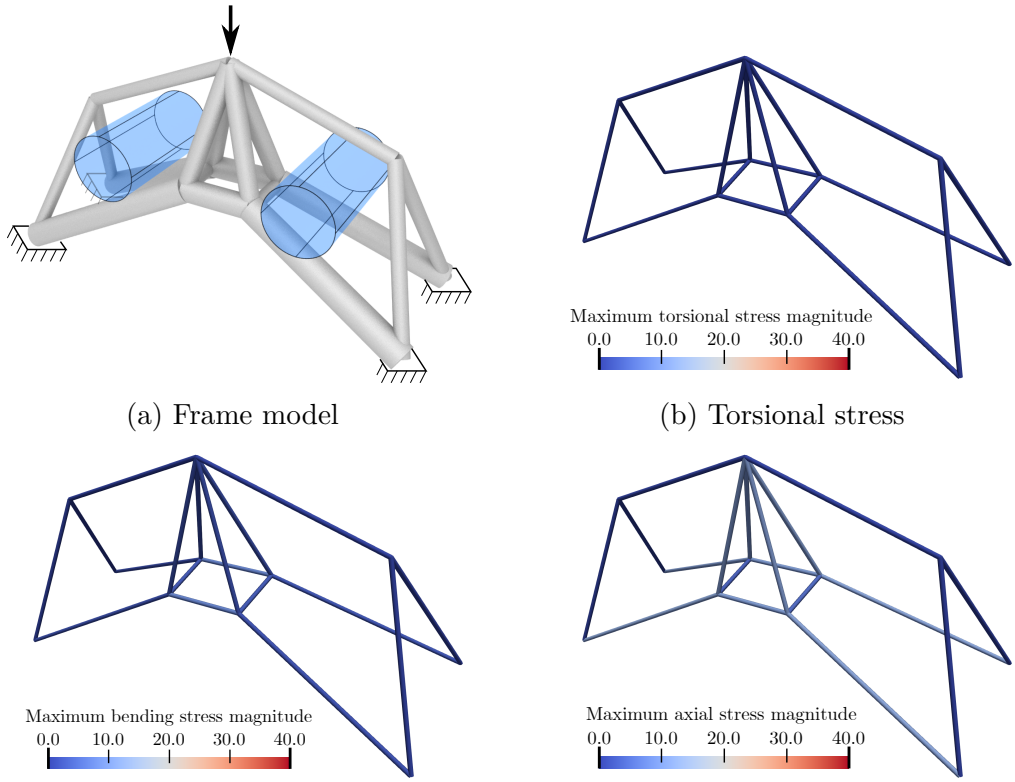
After three size optimisation and two layout optimisation steps, the frame shown in Figure 2.19a is obtained. Note that with the implicitly defined geometric constraints, all members in the final optimal frame are strictly constrained within the given design domain. Similar to the previous examples, the symmetry is recovered, e.g. node 2 and 6 are now symmetric about the line $y = 20.0$. This significantly reduces the magnitude and variance of the member stresses, especially torsional stresses. The torsion stresses in the final model become negligible, with a distribution of only 0.085628 ± 0.061505. The bending stresses are also significantly reduced, and their distribution is 1.9325 ± 2.0381 compared to their distribution in the initial frame of 11.243 ± 4.2349. As expected, the frame becomes stretch dominant with the axial stress distribution of 6.8794 ± 5.2755.



(a) Frame model (b) Torsional stress

(c) Bending stress (d) Axial stress

Fig. 2.18 Geometry and stresses of the initial 3D frame. Model (a) shows the actual cross-sectional sizes and blue transparent cylinders highlight the holes. (b) Torsional stress has mean 4.1508, variance 2.3121 and maximum 7.5895. (c) Bending stress has mean 11.243, variance 4.2349 and maximum 22.121. (d) Axial stress has mean 9.7061, variance 10.9358 and maximum 39.5428.

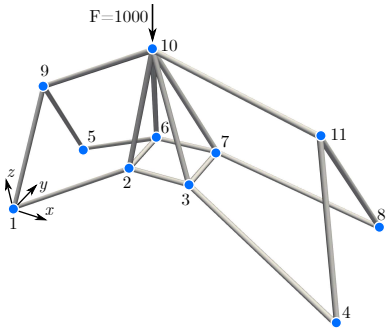


(a) Frame model (b) Torsional stress

(c) Bending stress (d) Axial stress

Fig. 2.19 Geometry and stresses of the final optimised 3D frame model. Model (a) shows the actual cross-sectional sizes and blue transparent cylinders highlight the holes. (b) Torsional stress has mean 0.085628, variance 0.061505 and maximum 0.23705. (c) Bending stress has mean 1.9325, variance 2.0381 and maximum 7.3246. (d) Axial stress has mean 6.8794, variance 5.2755 and maximum 12.055.

Table 2.9 Node coordinates of the optimal 3D frame



Node	x	y	z	Node	x	y	z
1	0.00	0.00	0.00	7	71.29	28.82	23.73
2	50.28	10.89	24.28	8	120.00	40.00	0.00
3	71.29	11.40	23.73	9	12.78	19.85	41.36
4	120.00	0.00	0.00	10	60.00	20.00	60.00
5	0.00	40.00	0.00	11	106.75	20.04	43.10
6	48.58	28.75	23.69				

The convergence of the optimisation along with the intermediate shapes are demonstrated in Figure 2.20. After five steps consisting of in total 164 sub-steps, the compliance starts from 21.281 and converges at 10.475 achieving a reduction of 50.78%.

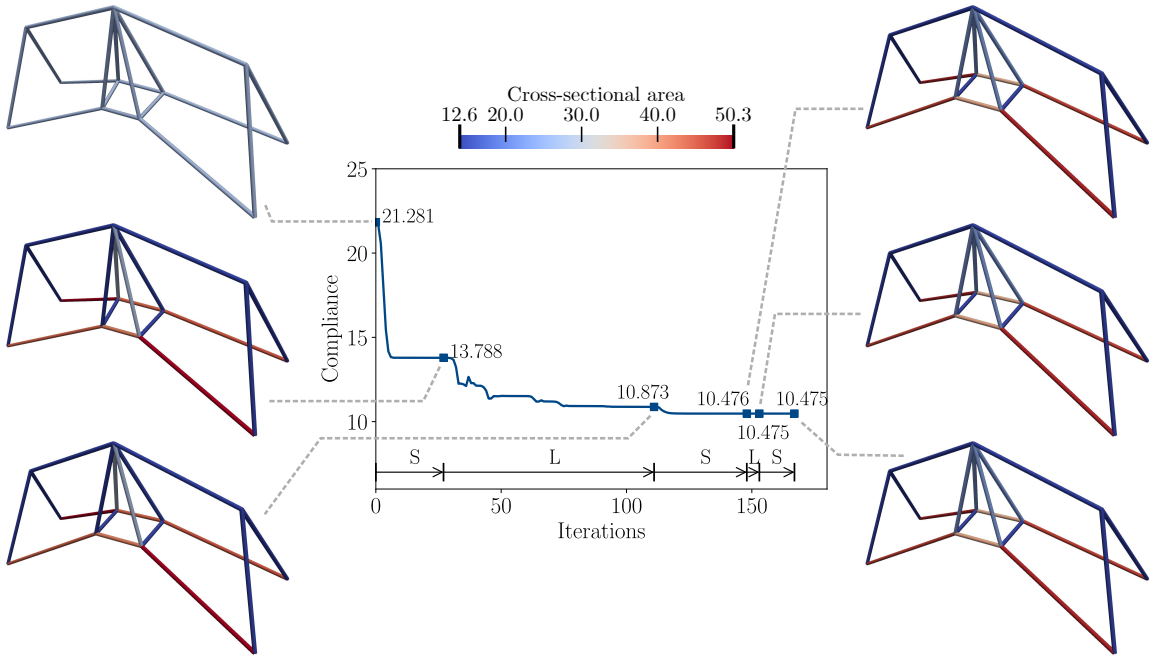


Fig. 2.20 Convergence of the compliance during the sequential size (S) and layout (L) optimisation of the 3D frame. It takes five steps to reach the minimum compliance and the numbers of sub-steps are $27(\mathrm{S}), 84(\mathrm{L}), 37(\mathrm{S}), 4(\mathrm{L})$ and $15(\mathrm{S})$. The layouts and member cross-sectional areas of intermediate structures during the optimisation are shown beside the convergence graph.

2.2.6 Limitations

It is worth mentioning that the final shape of the proposed frame optimisation process largely depends on the topology of the initial frame. The layout optimisation proposed here cannot add new members to the existing structure, i.e. it can only remove the short members. But if the initial frame has a structurally sound topology, the frame optimisation in this section can produce a faithful optimal shape with converged optimal compliance. In the overall approach proposed in this thesis, the frame model comes from the skeletonised topology-optimised geometry, which is usually a structurally sound model with all the critical load-paths present.

CHAPTER 3

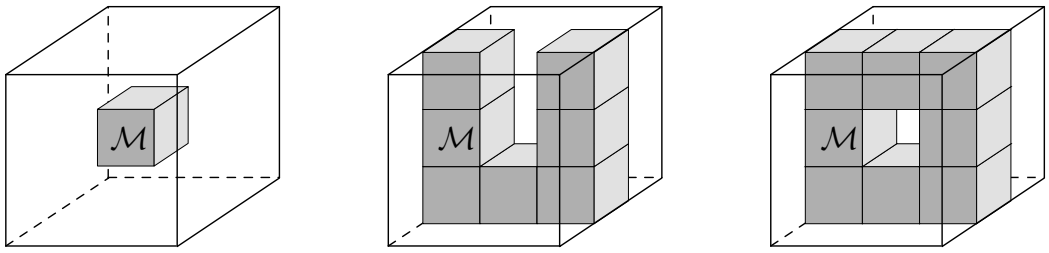
Digital topology and homotopic skeletonisation

This chapter introduces skeletonisation techniques and the concept of digital topology underlying them. Skeletonisation works as the essential component of the proposed CAD model generation process, which can significantly reduce the complexity of the geometric representation of the topology-optimised mesh while preserving the critical load path revealed by topology optimisation. The hexahedron meshes from topology optimisation are considered as 3D images so that mathematical and algorithmic tools in digital topology can be applied to analyse them. In order to understand the digital topology, Section 3.1 starts with an introduction of the core invariant in digital topology, named as Euler characteristic, in Section 3.1.1. Then Section 3.1.2 elaborates the mathematical foundation of topology conservation using Euler characteristic. A topology-preserving, or homotopic, skeletonisation technique is introduced in Section 3.2. Section 3.2.1 reviews related skeletonisation techniques and Section 3.2.2 details the use of the skeletonisation algorithm in structural and mechanical engineering applications. Lastly, in Section 3.3, several skeletonisation examples and a discussion of robustness and efficiency are provided.

3.1 Review of digital topology

Digital topology aims to classify a given 3D grid consisting of a set of *voxels*. A voxel, denoted as v, is the basic element in digital topology with the centroid (x_v, y_v, z_v). The volume mesh can be seen as a subset $\mathcal{M}$ in 3D integer space $\mathbb{Z}^3$. It consists of the corresponding topological entities (vertices, edges, faces and hexahedrons) belonging to

solid voxels. Solid voxels, in this thesis, are the hexahedrons in thresholded topology-optimised geometries. Accordingly, the voxels not in the volume mesh are *void voxels*. For example, as illustrated in Figure 3.1a, the volume mesh $\mathcal{M}$ has 8 vertices, 12 edges, 6 faces and 1 hexahedron. These belong to the one solid voxel in the image, whereas the $\mathcal{M}$ in Figure 3.1b has 32 vertices, 60 edges, 32 faces and 7 hexahedrons, and $\mathcal{M}$ in Figure 3.1c has 32 vertices, 64 edges, 40 faces and 8 hexahedrons.



(a) (b) (c)

Fig. 3.1 Volume meshes are seen as objects in 3D binary images. The 3D space is bounded with the black lines and the solid voxels are in grey. The object $\mathcal{M}$ in (a) has 1 voxel, the object $\mathcal{M}$ in (a) has 7 voxels and the object $\mathcal{M}$ in (a) has 8 voxels.

3.1.1 Euler characteristic

As a topological invariant, the *Euler characteristic* is the number that describes a shape or geometry regardless of the way it is bent [45]. Understanding of a volume mesh's topology is mainly acquired through tracing its Euler characteristic. The Euler characteristic of the volume mesh $\mathcal{M}$ can be determined as

$$\chi(\mathcal{M}) = n_0(\mathcal{M}) - n_1(\mathcal{M}) + n_2(\mathcal{M}) - n_3(\mathcal{M})\,, \tag{3.1}$$

and the Euler characteristic of a surface is

$$\chi(\partial\mathcal{M}) = n_0(\partial\mathcal{M}) - n_1(\partial\mathcal{M}) + n_2(\partial\mathcal{M})\,, \tag{3.2}$$

where $\partial\mathcal{M}$ is the bounded surface of the volume mesh $\mathcal{M}$, $n_0(\cdot)$, $n_1(\cdot)$, $n_2(\cdot)$ and $n_3(\cdot)$ denote the number of vertices, edges, faces and hexahedrons respectively in $\mathcal{M}$ and $\partial\mathcal{M}$.

For instance, for the volume mesh shown in Figure 3.1a, the Euler characteristic of it can be determined as $\chi(\mathcal{M}) = 8 - 12 + 6 - 1 = 1$ using (3.1), and its bounded surface as $\chi(\partial\mathcal{M}) = 8 - 12 + 6 = 2$ using (3.2). The Euler characteristic of this volume mesh is the same as that of a sphere. Therefore, topologically, the volume

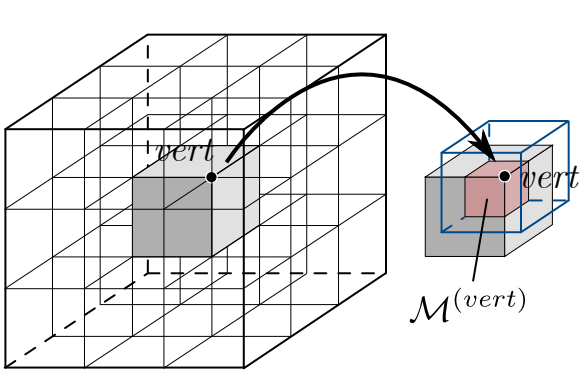
mesh in Figure 3.1a is the same as, or *homeomorphic* to, a sphere. The volume mesh in Figure 3.1b is also homeomorphic to a sphere, which has the Euler characteristics $\chi(\mathcal{M}) = 32-60+36-7 = 1$ and $\chi(\partial\mathcal{M}) = 32-60+30 = 2$. The Euler characteristics of the volume mesh in Figure 3.1c are calculated as $\chi(\mathcal{M}) = 32-64+40-8 = 0$ and $\chi(\partial\mathcal{M}) = 32-64+32 = 0$, which means it is homeomorphic to a 1-torus. Note that the relation

$$\chi(\mathcal{M}) = \frac{1}{2}\chi(\partial\mathcal{M}) \tag{3.3}$$

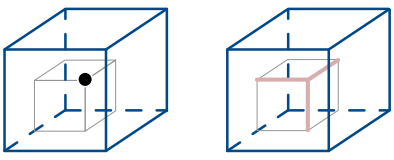
is always true in digital topology as proven by Lee et al. [44]. Because the Euler characteristic of a surface and a volume mesh can easily be converted to each other using relation (3.3), in the following discussion, only the Euler characteristic of volume mesh is considered.

a. Vertex window

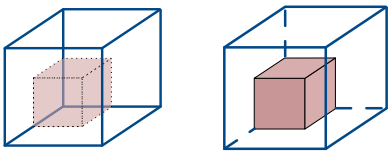
Since the Euler characteristic is additive [90], the Euler characteristic of a volume mesh can also be determined by summing local Euler characteristic contributions. As illustrated in Figure 3.2a, a one-voxel sized window, bounded by blue lines, is centred at a vertex named *vert*; here this window is called as the *vertex window* of *vert*. The volume mesh in this window is denoted as $\mathcal{M}^{(vert)}$, which is the pink object shown in Figure 3.2a. The Euler characteristic of a volume mesh can be determined by summing the Euler characteristic contributions of the objects in all vertex windows in this volume



(a) The volume mesh and the vertex window



(b) Vertices (c) Edges



(d) Faces (e) Hexahedrons

Fig. 3.2 The local Euler characteristic contribution from the vertex window. In The volume mesh in (a) consists of one voxel. The vertex window of *vert* is highlighted in blue in (a) and the objects in this window are in pink. Topological entities in $\mathcal{M}^{(vert)}$ are one vertex in (b), three half-edges in (c), three quarter-faces in (d) and one one-eighth-hexahedron in (e).

mesh, which reads

$$\chi(\mathcal{M}) = \sum_{vert \in \mathcal{M}} \chi(\mathcal{M}^{(vert)}) \,. \tag{3.4}$$

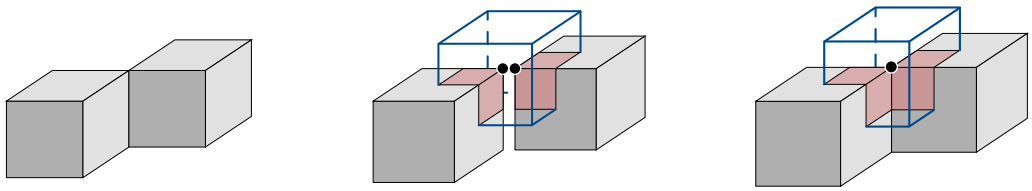
In order to determine the Euler characteristic of $\mathcal{M}^{(vert)}$ all topological entities in the vertex window should be correctly counted. In Figure 3.2a the counting of topological entities in the vertex window gives: one vertex, three half-edges, three quarter-faces and one one-eighth-hexahedron, see Figures 3.2b to 3.2e. As a result the Euler characteristic of $\mathcal{M}^{(vert)}$ contributed from these entities using (3.1) is calculated as $\chi(\mathcal{M}^{(vert)}) = 1 - \frac{3}{2} + \frac{3}{4} - \frac{1}{8} = \frac{1}{8}$. When summing the local contributions from all 8 vertex windows, the Euler characteristic of the volume mesh in Figure 3.2a is $\chi(\mathcal{M}) = \frac{1}{8} \times 8 = 1$, which is identical to the result if the Euler characteristic is computed on the whole volume mesh. In general, the Euler characteristic of a volume mesh $\mathcal{M}$ is determined as

$$\chi(\mathcal{M}) = \sum_{vert \in \mathcal{M}} \left(n_0^{(vert)} - \frac{n_1^{(vert)}}{2} + \frac{n_2^{(vert)}}{4} - \frac{n_3^{(vert)}}{8} \right), \tag{3.5}$$

where $n_0^{(vert)}$, $n_1^{(vert)}$, $n_2^{(vert)}$ and $n_3^{(vert)}$ are the number of vertices, half-edges, quarter-faces and one-eighth-hexahedrons in $\mathcal{M}^{(vert)}$.

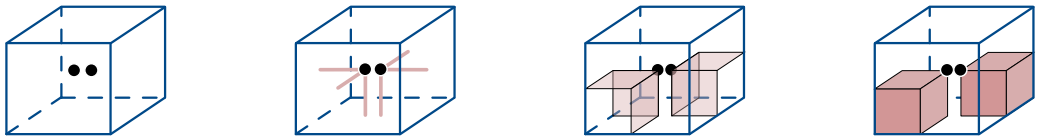
b. Neighbourhood

Considering the case where $\mathcal{M}^{(vert)}$ is associated with more than one voxel, the connectivity of voxels needs to be considered. A volume mesh containing two voxels is shown in Figure 3.3a. If these two voxels are regarded as two separate objects shown in 3.3b, then the topological entities in $\mathcal{M}^{(vert)}$ are two vertices, six half-edges, six quarter-faces and two one-eighth-hexahedrons, see Figure 3.4. This leads to the Euler characteristic of the objects in the vertex window being $2 - \frac{6}{2} + \frac{6}{4} - \frac{2}{8} = \frac{1}{4}$ using (3.5).



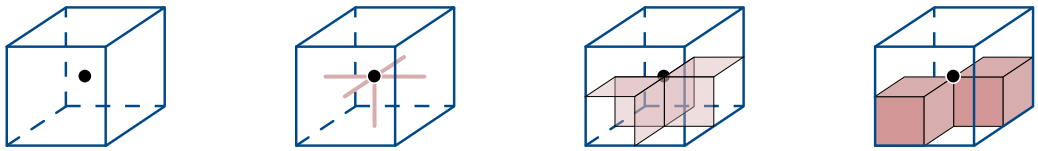
(a) (b) (c)

Fig. 3.3 Different neighbourhood definitions lead to different topological entries. The two voxels of the object can be visualised in (b) as separate or in (c) as connected. The gap between two separate voxels in (b) is exaggerated.



(a) Vertex entities (b) Edge entities (c) Face entities (d) Volume entities

Fig. 3.4 Topological entries for two voxels regarded as separated



(a) Vertex entities (b) Edge entities (c) Face entities (d) Volume entities

Fig. 3.5 Topological entries for two voxels regarded as connected

Otherwise, in Figure 3.3c, if these two voxels are connected, then the topological entities in the vertex window are one vertex, five half-edges, six quarter-faces and two one-eighth-hexahedrons, see Figure 3.5. As a result, the Euler characteristic of the objects in the vertex window is calculated as $1 - \frac{5}{2} + \frac{6}{4} - \frac{2}{8} = -\frac{1}{4}$. Evidently, this Euler characteristic is different from that determined when considering two voxels are connected. Therefore, the use of (3.5) requires an unambiguous definition of the *neighbourhood*. The neighbourhood of a voxel v is a set of voxels connected to v. Three widely-used neighbourhoods in digital topology are *6-neighbourhood*, *18-neighbourhood* and *26-neighbourhood* respectively [45, 43]. With the voxel size of $h \in \mathbb{Z}^+$, these three neighbourhoods of the voxel v centred at (x_v, y_v, z_v) are defined as

(i) 6-neighbourhood (see Figure 3.6a):

$$\mathcal{N}_6(v) = \{w \,|\, |x_v - x_w| + |y_v - y_w| + |z_v - z_w| \leq h\} \ , \tag{3.6}$$

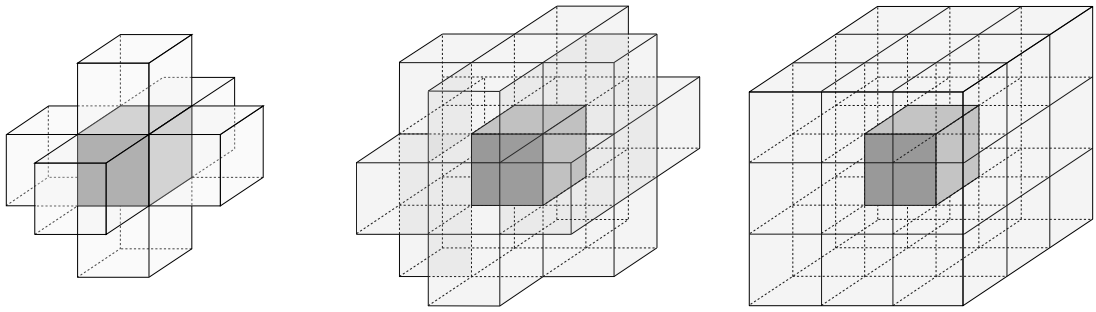
(ii) 18-neighbourhood (see Figure 3.6b):

$$\mathcal{N}_{18}(v) = \{w \,|\, |x_v - x_w| + |y_v - y_w| + |z_v - z_w| \leq 2h\} \ , \text{and} \tag{3.7}$$

(iii) 26-neighbourhood (see Figure 3.6c):

$$\mathcal{N}_{26}(v) = \{w \,|\, \max(|x_v - x_w|, |y_v - y_w|, |z_v - z_w|) \leq h\}. \tag{3.8}$$

In this thesis, solid voxels are considered in 26-neighbourhood, whereas any two void voxels are considered to be neighbours when they are 6-neighbours. It is important to

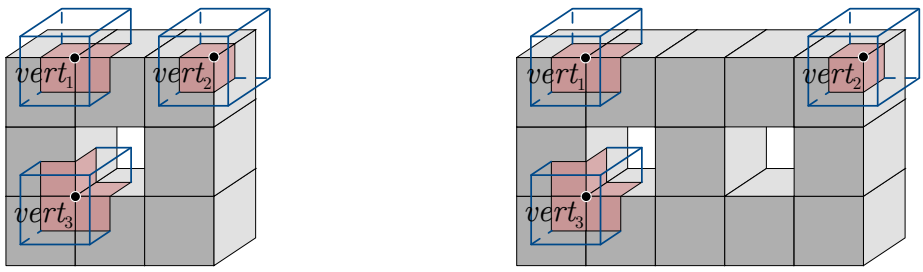


(a) 6-neighbourhood (b) 18-neighbourhood (c) 26-neighbourhood

Fig. 3.6 Three neighbourhood definitions of voxels on uniform grids. All possible positions of neighboured voxels to the centred opaque grey voxel are coloured in transparent grey.

use compatible neighbourhood definitions for both the solid and void voxels. The use of a single neighbourhood definition leads to ambiguities, for instance, in the planar case to the violation of the *Jordan curve theorem* [46]. The Jordan curve theorem states that a simple closed curve divides the plane into an interior and an exterior region. Note that the neighbourhoods (3.6), (3.7) and (3.8) are not suitable for non-uniform grids. In this thesis, only consider uniform grids are considered. For the interested reader, the neighbourhood defined on non-uniform grids can be found in [91].

Considering the volume mesh in Figure 3.7a as an example, if 26-neighbourhood is used, the Euler characteristics of the objects in the vertex windows of $vert_1$, $vert_2$ and $vert_3$ can be calculated as: $\chi(\mathcal{M}^{(vert_1)}) = 1-\frac{4}{2}+\frac{5}{4}-\frac{2}{8} = 0$, $\chi(\mathcal{M}^{(vert_2)}) = 1-\frac{3}{2}+\frac{3}{4}-\frac{1}{8} = \frac{1}{8}$ and $\chi(\mathcal{M}^{(vert_3)}) = 1-\frac{5}{2}+\frac{7}{4}-\frac{3}{8} = -\frac{1}{8}$ respectively. Considering the symmetry, 16 vertex windows contain the same objects as $\mathcal{M}^{(vert_1)}$, 8 vertex windows as $\mathcal{M}^{(vert_2)}$ and 8 vertex windows as $\mathcal{M}^{(vert_3)}$. Therefore, the Euler characteristic of the volume mesh in Figure



(a) (b)

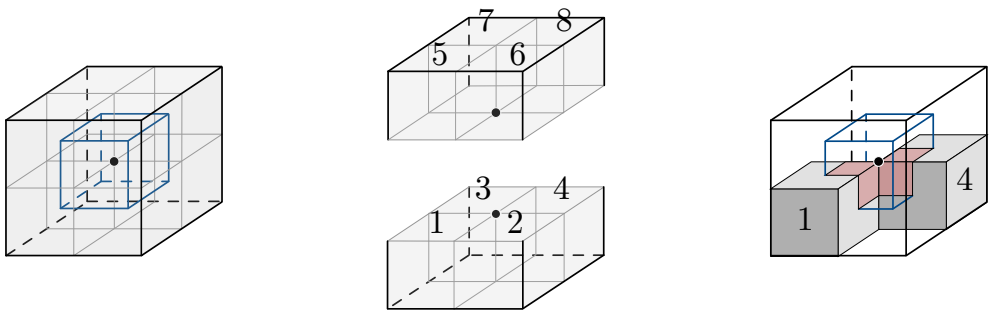
Fig. 3.7 Vertex-wise approach to determine Euler characteristics of two meshes. The vertex windows are highlighted in blue. Based on the Euler characteristics, (a) is homeomorphic to a 1-torus, whereas (b) is homeomorphic to a 2-torus.

3.7a is $\chi(\mathcal{M}) = 16\chi(\mathcal{M}^{(vert_1)})+8\chi(\mathcal{M}^{(vert_2)})+8\chi(\mathcal{M}^{(vert_3)}) = 16\times0+8\times\frac{1}{8}+8\times(-\frac{1}{8}) = 0$. Similarly, the Euler characteristic of the volume mesh in Figure 3.7b is $\chi(\mathcal{M}) = 20\chi(\mathcal{M}^{(vert_1)}) + 8\chi(\mathcal{M}^{(vert_2)}) + 16\chi(\mathcal{M}^{(vert_3)}) = 20 \times 0 + 8 \times \frac{1}{8} + 16 \times (-\frac{1}{8}) = -1$.

c. Octant

The contribution of the object in a vertex window is always associated with eight voxels. As shown in Figure 3.8a, the vertex window of the black point overlaps eight voxels. The set of these $2 \times 2 \times 2$ voxels is an *octant* centred at the vertex *vert*, denoted as $Oct(vert)$. Note that even though for convenience the configuration of a octant is described using the solid-void states of the associated eight voxels, the objects included in $Oct(vert)$ are identical to the objects in $\mathcal{M}^{(vert)}$.

The contribution of each octant, $\chi(Oct(vert))$, can be precomputed and stored in a look-up table. The eight voxels in an octant have $2^8 = 256$ possible solid-void states and considering symmetries this reduces to only 22 distinct cases. Tables with contributions of octants to the Euler characteristic can be found, for instance, in [44, 43] and also in Table D.1. When using these tables, one needs an 8-bit binary code for the configuration, such as 00001001 for the octant shown in Figure 3.8c with the numbering illustrated in Figure 3.8b. The 1s in this binary code mark the position indices of the solid voxels.



(a) (b) (c)

Fig. 3.8 Octant and its binary configuration code. The blue window in (a) represents the domain of the vertex window of the black point. (b) gives the numbering for the eight associated voxels in the vertex window. The vertex window in (c) has the binary code of 00001001, which has $\chi = -\frac{1}{4}$ according to Table D.1.

d. Voxel window

The Euler characteristic of the volume mesh may change when new topological entities are introduced or current ones are removed. Determining the Euler characteristic of the

volume mesh to trace these changes is computationally expensive. Instead, as suggested in [46, 44], the effects on the Euler characteristic of a volume mesh corresponding to its topological entity change can be investigated within a *voxel window*. A voxel window, denoted as $\mathcal{M}(\mathcal{N}_{26}(v))$, is centred at v and of a size 1.5 times the voxel size. The voxels overlapping it are considered in 26-neighbourhoods. As illustrated in Figure 3.9a, the blue window is the voxel window of the green voxel. The objects in $\mathcal{M}(\mathcal{N}_{26}(v))$ are the union of the octants centred at eight vertices of v, see Figures 3.9b to 3.9i, which reads

$$\mathcal{M}(\mathcal{N}_{26}(v)) = \bigcup_{vert \in v} Oct(vert)\,. \tag{3.9}$$

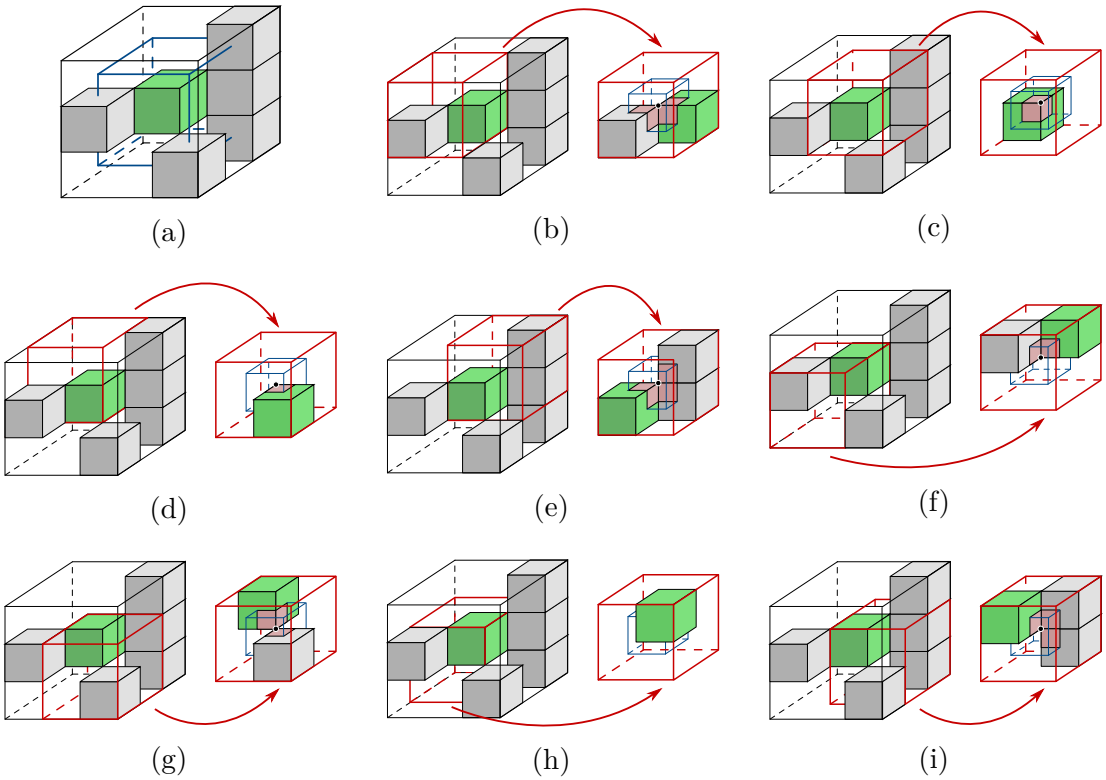


Fig. 3.9 The eight octants associated with the green voxel v. The objects in the blue window in (a) are $\mathcal{M}(\mathcal{N}_{26}(v))$. From (b) to (i), $\mathcal{M}(\mathcal{N}_{26}(v))$ is split into eight octants highlighted in red bounded boxes. These eight octants overlap voxel v and each of them centres at one of the eight vertices of the green voxel. They have Euler characteristics of (b) $-\frac{1}{4}$, (c) $\frac{1}{8}$, (d) $\frac{1}{8}$, (e) $-\frac{3}{8}$, (f) $-\frac{3}{4}$, (g) $-\frac{3}{4}$, (h) $-\frac{1}{8}$ and (i) $-\frac{3}{8}$ respectively.

Accordingly, the Euler characteristic of $\mathcal{M}(\mathcal{N}_{26}(v))$ is the sum of the Euler characteristic contributions from the eight octants as

$$\chi(\mathcal{M}(\mathcal{N}_{26}(v))) = \sum_{vert \in v} \chi(Oct(vert)) \,. \tag{3.10}$$

When using (3.10), one can simply refer to the look-up table, e.g. Table D.1, to obtain the Euler characteristic of each octant. Normally, the voxel of interest is placed at position 1 in the numbering, see Figure 3.8b. As illustrated in Figure 3.9, adding together the Euler characteristics of 8 octants associated with the green voxel v, from Figure 3.9b to 3.9i, gives the Euler characteristics of the union of objects in the blue window. The Euler characteristic of each octant can be easily found according to its binary configuration code in the look-up table. As a result, the Euler characteristic contribution from objects in $\mathcal{M}(\mathcal{N}_{26}(v))$ is calculated as $-\frac{1}{4}+\frac{1}{8}+\frac{1}{8}-\frac{3}{8}-\frac{3}{4}-\frac{3}{4}+\frac{1}{8}-\frac{3}{8}=-\frac{17}{8}$, where v is the green voxel of interest.

3.1.2 Topology preservation

The Euler characteristic of a more complex voxel model is related to the total number of separate objects (connected components) $O(\mathcal{M})$, holes (handles) $H(\mathcal{M})$ and cavities $C(\mathcal{M})$ in the entire model

$$\chi(\mathcal{M}) = O(\mathcal{M}) - H(\mathcal{M}) + C(\mathcal{M}). \tag{3.11}$$

The Euler characteristic is critical in determining the voxels at the boundary of the solid mesh that can be deleted without changing its topology. These can-be-deleted voxels are *simple points*. The simple point is the voxel such that the deletion of this voxel does not lead to a change in topology [46]. As evident from (3.11) simply conserving the Euler characteristic $\chi(\mathcal{M})$ of the entire or a portion of the mesh does not ensure that the topology, or the number of objects, tunnels and cavities, remains the same. As a result, for a solid border voxel v to be classified as simple, its deletion must not change the number of objects and holes for both $\mathcal{M}$ and its complement set [46]. According to Lee et al. [44], these conditions can be checked by examining the state of the voxels in the 26-neigbourhood $\mathcal{N}_{26}(v)$ of a voxel v, which contains all the octants overlapping v.

That is, a border voxel v is a simple point if and only if

$$\Delta\chi(\mathcal{M}(\mathcal{N}_{26}(v))) = 0 \quad \text{and} \tag{3.12a}$$
$$\Delta O(\mathcal{M}(\mathcal{N}_{26}(v))) = 0 \quad \text{or} \tag{3.12b}$$
$$\Delta H(\mathcal{M}(\mathcal{N}_{26}(v))) = 0\,, \tag{3.12c}$$

where Δ denotes the change in the respective quantities with and without voxel v present. Note that the *Euler invariant* is the voxel such that deleting this voxel does not change the Euler characteristic of the volume mesh, i.e. (3.12a) is held.

a. Check for the change in Euler characteristic

It is evidently straightforward to compute the change in Euler characteristic (3.12a) using (3.10) and Table D.1. In each octant, the change in Euler characteristic can be determined by subtracting $\chi(\mathcal{M}(\mathcal{N}_{26}(v)))$ from $\chi(\mathcal{M}(\mathcal{N}_{26}(v)) \setminus v)$. The sum of these changes gives the change in Euler characteristic, caused by deleting v, of the objects in the voxel window of v, which reads

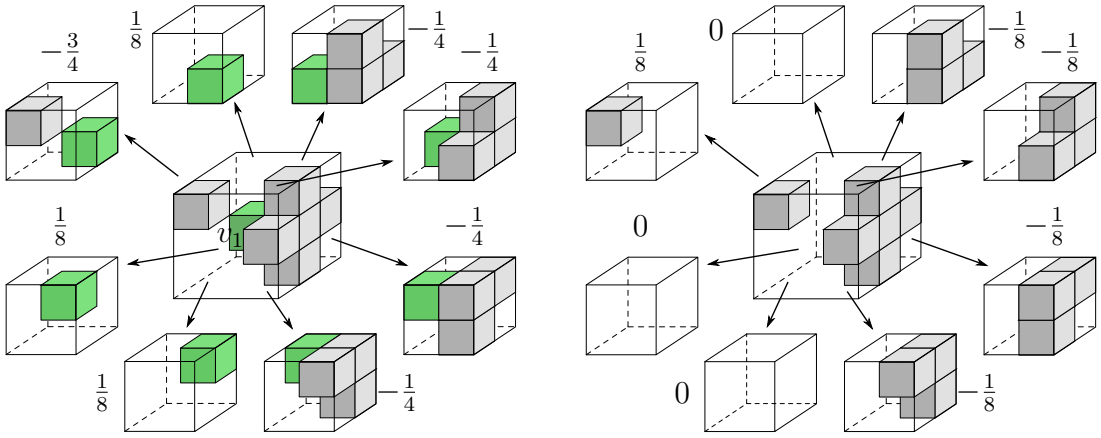
$$\begin{aligned}\Delta\chi(\mathcal{M}(\mathcal{N}_{26}(v))) &= \chi(\mathcal{M}(\mathcal{N}_{26}(v) \setminus v)) - \chi(\mathcal{M}(\mathcal{N}_{26}(v))) \\ &= \sum_{vert \in v} (\chi(Oct(vert) \setminus v) - \chi(Oct(vert))).\end{aligned} \tag{3.13}$$

For instance, as shown in Figure 3.10a, the change in Euler characteristic due to the deletion of the green voxel v_1 is given using (3.13) as

$$\begin{aligned}\Delta\chi(\mathcal{M}(\mathcal{N}_{26}(v_1))) &= \sum_{vert \in v_1} (\chi(Oct(vert) \setminus v_1) - \chi(Oct(vert))) \\ &= \left(\frac{1}{8} - (-\frac{3}{4})\right) + \left(0 - \frac{1}{8}\right) + \left(-\frac{1}{8} - (-\frac{1}{4})\right) + \left(-\frac{1}{8} - (-\frac{1}{4})\right) \\ &\quad + \left(-\frac{1}{8} - (-\frac{1}{4})\right) + \left(-\frac{1}{8} - (-\frac{1}{4})\right) + \left(0 - \frac{1}{8}\right) + \left(0 - \frac{1}{8}\right) \\ &= 1 \neq 0\,.\end{aligned}$$

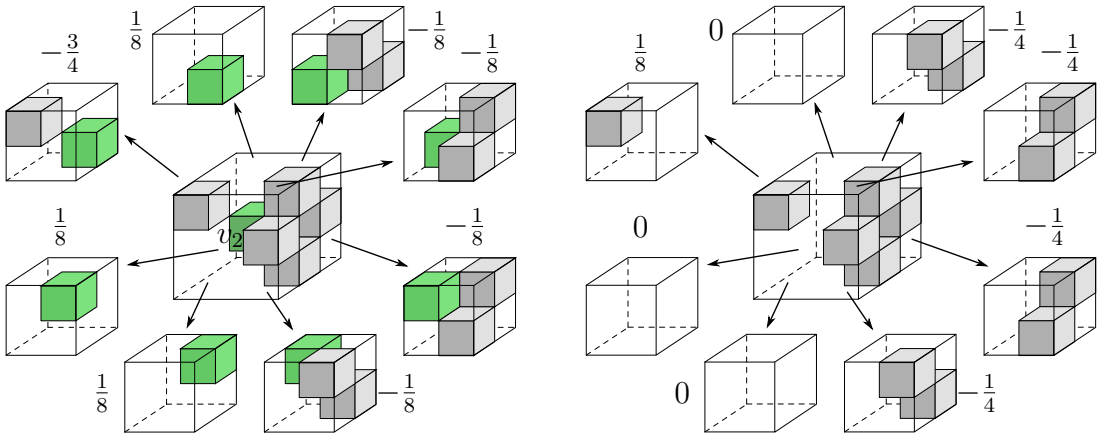
With $\Delta\chi(\mathcal{M}(\mathcal{N}_{26}(v_1))) \neq 0$, deleting voxel v_1 changes the topology of the volume mesh.

For convenience, a table of change in Euler characteristic $\Delta\chi(Oct(vert))$ and the corresponding configuration of $Oct(vert)$ is recorded in [44, 43] and Table D.2. The voxel-to-delete is placed at the last digit of the binary code, i.e. the position 1 in Figure 3.8b.



(a) $\mathcal{N}_{26}(v_1)$ (b) $\mathcal{N}_{26}(v_1) \setminus v_1$

Fig. 3.10 Octants and their Euler characteristics in $\mathcal{N}_{26}(v_1)$ and $\mathcal{N}_{26}(v_1) \setminus v_1$. (a) shows the eight octants associated with v_1 before the removal of v_1, and the deletion of v_1 changes them into eight octants in (b). The Euler characteristic of each octant is shown around the octant. Visually the topology changes due to the deletion, i.e. there are two objects in (b) while only one object in (a); this is further confirmed by the change in Euler characteristics between $\mathcal{N}_{26}(v_1)$ and $\mathcal{N}_{26}(v_1) \setminus v_1$.



(a) $\mathcal{N}_{26}(v_2)$ (b) $\mathcal{N}_{26}(v_2) \setminus v_2$

Fig. 3.11 Octants and their Euler characteristics in $\mathcal{N}_{26}(v_2)$ and $\mathcal{N}_{26}(v_2) \setminus v_2$. (a) shows the eight octants associated with v_2 before the removal of v_2, and eight octants without v_2 are shown in (b). The Euler characteristic of each octant is shown around the octant. As visually evident, the deletion causes the change in topology; $\mathcal{M}(\mathcal{N}_{26}(v_2))$ has one object while $\mathcal{M}(\mathcal{N}_{26}(v_2)) \setminus v_2$ has two objects and one hole. However, according to (3.11), the Euler characteristic will remain the same.

As another example shown in Figure 3.11, the change in Euler characteristic reads

$$\begin{aligned}\Delta\chi(\mathcal{M}(\mathcal{N}_{26}(v_2))) &= \sum_{vert\in v_2}(\chi(Oct(vert)\setminus v_2)-\chi(Oct(vert)))\\ &=\left(\frac{1}{8}-(-\frac{3}{4})\right)+\left(0-\frac{1}{8}\right)+\left(-\frac{1}{4}-(-\frac{1}{8})\right)+\left(-\frac{1}{4}-(-\frac{1}{8})\right)\\ &\quad+\left(-\frac{1}{4}-(-\frac{1}{8})\right)+\left(-\frac{1}{4}-(-\frac{1}{8})\right)+\left(0-\frac{1}{8}\right)+\left(0-\frac{1}{8}\right)\\ &=0.\end{aligned}$$

The calculation above shows the deletion of the green voxel v_2 does not change the Euler characteristic of the volume mesh, i.e. v_2 is an Euler invariant. However, the removal of v_2 leads to breaking one connected object into two separate objects, which violates the condition (3.12b) for topology preservation.

b. Check for the number of connected objects

The example in Figure 3.11b shows the necessity of checking (3.12b) or (3.12c) in topology preservation. It is relatively easier to determine the number of objects using (3.12b) in $\mathcal{M}(\mathcal{N}_{26}(v))$ than the number of holes in (3.12c). Lee et al. [44] state that for a solid voxel v, $\Delta O(\mathcal{M}(\mathcal{N}_{26}(v)))=0$ is held if and only if $O(\mathcal{M}(\mathcal{N}_{26}(v))\setminus v)=1$. One can use a recursive algorithm and an octree data structure introduced in [44] to determine the number of the objects in $\mathcal{M}(\mathcal{N}_{26}(v))\setminus v$. As an alternative, one can convert the connected voxels in their 26-neighbourhood into a undirected graph[1] and simply use *breadth first search* (BFS) or *depth first search* (DFS) [92] to count the number of connected components.

The green voxel v_2 shown in Figure 3.11a, despite being checked as an Euler invariant, cannot be deleted because the connected objects in the configuration after the deletion is 2, see Figure 3.11b. In this case, v_2 in Figure 3.11a is not a simple point[2].

3.2 Skeletonisation algorithm

Topological skeletons include curve skeletons and surface skeletons. These are also known as medial axis and medial surface, respectively. The process of determining a topological skeleton is called skeletonisation. Curve and surface skeletonisation are

[1] Not to be confused with the graph model used in the skeleton-frame conversion.
[2] See (3.12).

also referred to as *Media Axis Transform* (MAT) and *Media Surface Transform* (MST) respectively. The skeletonisation process can take either surface or volume meshes as inputs, and reduces their dimensionality into curve and surface skeleton as output.

3.2.1 Review of skeletonisation approaches

Skeletonisation techniques are widely used in object identification, shape analysis, pattern recognition and image segmentation [39]. The relevant research is active in the fields of medical imaging, animation industry, structural and mechanical engineering.

a. Definition of topological skeleton

The definition of topological skeleton varies in the different skeletonisation approaches. For instance, Blum [93] gave the definition of a topological skeleton as the locus of the centres of all maximal inscribed hyper-spheres, while in *grassfire analogy* [94], the skeleton can also be the loci where the fire fronts meet and quench each other. Summarising early years of research [36–38, 95, 96], the topological skeleton is defined using its major properties as listed below.

- **Homotopy** The skeleton is homeomorphic to its formal counterpart.
- **Thinness** The curve skeleton is one-dimensional, and the surface skeleton is two-dimensional.
- **Centredness** The curve skeleton lies on the medial surface, and the surface skeleton is centred within the object. Instead of exact centredness, an approximate centredness is sufficient for many applications.

Along with the properties above, researchers also consider other properties, e.g. the reliability, constructibility and invariance under isometric transformations; the reader can refer to [37, 38] for details.

b. Skeletonisation of surface meshes

For surface meshes, Ogniewicz [97] used Voronoi diagrams to compute their polygonal skeletons. However, because the skeleton is sensitive to mesh boundary, early-stage of Voronoi diagram approaches are not suitable for skeletonising objects with complex boundaries [98]. Later research developed less boundary-sensitive skeletonisation processes using Voronoi diagrams [98–101]. Among these methods, those using Voronoi balls, or known as *Power Crust* [102, 103], generally give better skeletons.

In parallel with the Voronoi diagram families, edge/mesh collapse is also a promising approach to skeletonising the surface meshes. This contraction-based skeletonisation technique was first developed in [104] by Li et al., who were inspired by the shape decomposition used in computer vision. They [104] claimed that in order to understand the topology and to acquire a sufficiently smooth skeleton, the geometric features of the mesh need to be well studied before the skeletonisation. Follow-up works focused on feature recognition, e.g. Katz and Tal [105] started the skeletonisation with clustering segmentations, Mortara et al. [106] firstly recognised different features of objects (tube or body) then used sphere sweeping to skeletonise the tube parts, and other related works can be seen in [49, 107, 108]. Researchers [109–113] also explored the possibility in skeletonising point clouds, since point clouds can be reconstructed as surface meshes. As mesh skeletonisation started to be considered for larger scale meshes, some supportive studies [110] have been conducted on speeding up the skeletonisation process using parallel computing techniques and GPUs.

c. Skeletonisation of volume meshes

For volumetric implicit representations, skeleton can be extracted using the *distance-field map* approaches. In these approaches, the pixels/voxels are assigned values which represent their Euclidean distances or other user-defined distances to the closest boundary surfaces. Such grid of cells with values is called the distance-field map. The method seeks the skeleton through singularities of borders in distance-field map, which are later identified as the joints of the curve skeleton [114]. Research has also been carried out in improving the robustness and smoothness of the resulting skeleton from distance-field maps [95, 115–120]. In general, distance-field map approaches produce a skeleton showing reliable centredness, but they cannot guarantee topology preservation during the process. Malandain and Fernández-Vidal [116] combined the distance-field map and the concept of simple point (as introduced in Section 3.1.2), to secure the homotopy of the process. There are also studies on skeletonisation particularly for level-set geometries [121, 122].

Researchers suggested the values assigned to pixels/voxels are not limited to Euclidean distances. Ahuja and Chuang [123] introduced 2D *potential field*. They assume the border carries electric charges, and the minima in the corresponding potential field within the domain gives the skeleton. This idea is extended into the 3D potential field by Ma and Wu [124]. Inspired by this, researchers started to use other physical systems to find medial axis, such as force vector method [125] and gradient flow vector method [126].

Another approach is the 3D thinning algorithm. This approach was first developed by Rosenfeld [90], who was inspired by very early research on pattern recognition [127]. He gave the framework of thinning algorithm for 2D images, and he also provided the insight of using symmetric removal of voxels to enforce the remaining voxels to locate in the centre of geometry. Later, Lee et al. [43] developed the 3D digital topology based on 2D studies conducted by Rosenfeld. The key to topology preservation is the identification of simple points. Discussion of the simple point can be found in the literature [44, 117, 128–130]. There are also supplementaries to the definition of simple point, such as *simple cell* [131], which extends the concept of simple point to non-hexahedrons.

In general, the 3D thinning algorithm proceeds through layer by layer erosion, and at each removal, only the border simple points are deleted. The homotopy of the process is guaranteed by checking Euler characteristics and connected objects. The centredness is enforced by sequential symmetric removals on border surfaces. The 3D thinning algorithm is chosen in this thesis because its superior robustness merits the solid fundamentals of digital topology and its high computational efficiency (there is a large number of libraries for 3D thinning algorithm designed using parallel computing).

In the following context, skeletonisation is specifically referred to as the skeletonisation process using 3D thinning algorithm.

3.2.2 Implementation details

The 3D thinning algorithm proceeds by removing the voxel which does not change the topology of the model based on the topology preservation criteria (3.12). Additionally, one removal step is split into six sub-steps and in each sub-step only the voxels approaching from one of the six grid directions are removed. This is necessary to preserve the geometric symmetry.

a. Curve skeletonisation

To start with, the 3D thinning algorithm for extracting the medial axis is given in Algorithm 3.1. The terminologies used in the Algorithm 3.1 is explained as follows.

[1]**Deletion direction** ensures that the skeleton can be extracted by symmetric deletions. Besides, there are many situations where simultaneously removal will cause complete elimination of the original object [44]. Each step of the skeletonisation process is divided into six sub-steps according to six grid directions, or principle coordinate

Algorithm 3.1: 3D thinning algorithm

Input : ($\mathcal{M}$), the volume mesh
Output : $Sk(\mathcal{M})$: the skeleton of $\mathcal{M}$

```
1  D: list of deletable candidates
2  while D ≠ ∅ in any one of the six deletion directions [1] do
3      foreach deletion direction ∈ {+x, −x, +y, −y, +z, −z} do
4          D ← ∅                                   ▷ initialise the deletion list
           /* First loop over the whole mesh to seek the deletable voxels   */
5          for v ∈ M do
6              if v is border voxel [2] and v is not tagged voxel [3] and v is not end
                voxel [4] and v is simple point [5] then
7                  insert v into D   ▷ v is checked as untagged non-end border simple point
           /* Second loop over deletable voxel list to do the double check  */
8          if D ≠ ∅ then
9              for w ∈ D do
10                 if w is simple point then
11                     delete w from M
12 return M as Sk(M)
```

directions. In this thesis, the sequence of $\{+x, -x, +y, -y, +z, -z\}$ is used, see Figure 3.12. The geometry of the resulting skeleton depends somewhat on the sequencing of these sub-steps, especially when the shape is simple. Here, the order of the sequence is not critical because (i) normally structural optimisation does not give simple shapes and (ii) tagged voxels are sued to secure the preservation of important features.

[2]**Border voxel** is the voxel on the outer boundary against a specific deletion direction. For example, in Figure 3.12a, where all border voxels corresponding to the deletion direction $+x$ are coloured in green, these border voxels are the ones with no neighboured voxels on $-x$ direction.

[3]**Tagged voxel** is user-defined. It is straightforward to tag voxels as non-removable if required. For instance, in finite element methods and structural optimisations, the voxels at Dirichlet and non-homogeneous Neumann boundaries are tagged as non-removable to preserve the structural meanings of certain voxels; or in order to keep specific geometric features, the voxels related to those features can also be set as tagged.

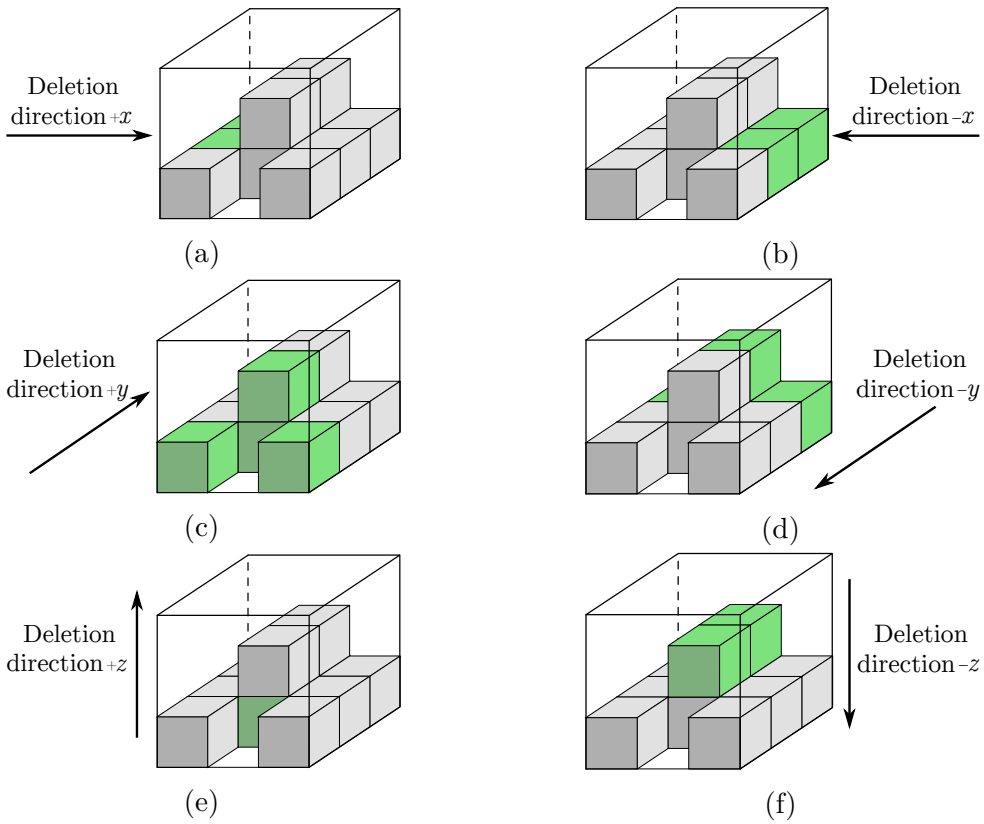


Fig. 3.12 The six deletion directions and corresponding border voxels. The border voxels are highlighted in green.

[4]**End voxel** is the voxel with only one neighboured voxel. It also marks the position where the one-voxel thick medial axis starts or ends .

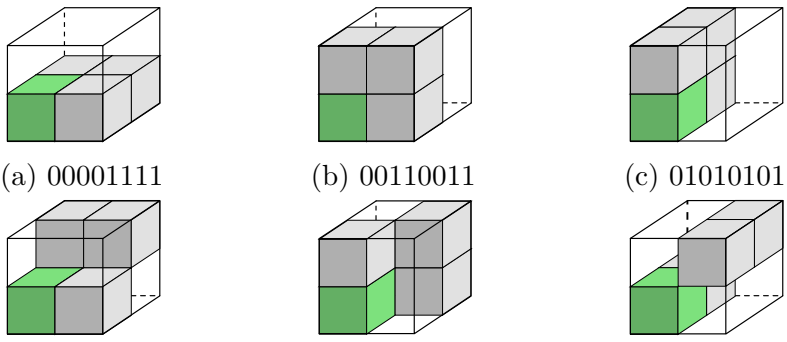
[5]**Simple point** is the voxel such that the removal of this voxel does not change the number of connected objects, cavities, and holes, i.e. (3.12) is held [132, 116, 128, 133].

The 3D thinning algorithm first detects and stores all simple untagged non-end border voxels into a candidate list $\mathcal{D}$. A sequential re-checking method is then used to ensure that connectivity is preserved even after the deletion of neighboured voxels, as suggested in [44, 134]. The recheck is necessary because the removal of voxels changes the mesh, albeit some simple points become 'non-simple'. The skeletonisation terminates when none of the remaining voxels is removable without violating condition (3.12) consecutively in all six deletion directions.

b. Surface skeletonisation

In some cases, medial surfaces rather than medial axes are desired to be kept, such as in shell and plate structures. Similar to curve skeletonisation, surface skeletonisation is adopted to acquire surfaces that are approximately located at the centre of the objects. To do so, one just simply alters the condition in line 6 of Algorithm 3.1 to 'v is border voxel and v is not tagged voxel and v is not *surface point* and v is simple point'. In the medial axis extraction, the end voxel is detected as the end of the voxel chain to ensure the mesh can be thinned into a one-voxel-thick chain. Similarly, the definition of a *surface point* is needed to detect the voxels on the edges of medial surfaces, so that the algorithm is able to skeletonise a given volume mesh to a one-voxel-thick skeleton in at least one of the three principal directions.

To define a surface point, two kinds of octants, see Figures 3.13a and 3.13d, and their symmetric rotations are considered, see Figures 3.13b, 3.13c, 3.13e and 3.13f. Using these configurations, voxels on a complete thin plane can be detected. Within one octant, a complete plane indicates the one-voxel thick plane formed by four voxels aligned as a one-voxel-thick plane. As can be seen in Figure 3.14a, 3.14b and 3.14c, all eight octants associated with the green voxel fall into one out of the six configurations in Figures 3.14a to 3.14f. As suggested in [44], in order to preserve the shape of the medial surface, the green voxels in Figures 3.14d, 3.14e and 3.14f on incomplete surface edges should also be preserved.



(a) 00001111 (b) 00110011 (c) 01010101

(d) 11000011 (e) 10011001 (f) 10100101

Fig. 3.13 Octant configurations for a voxel on a complete plane. The configuration codes are given using the numbering in Figure 3.8b.

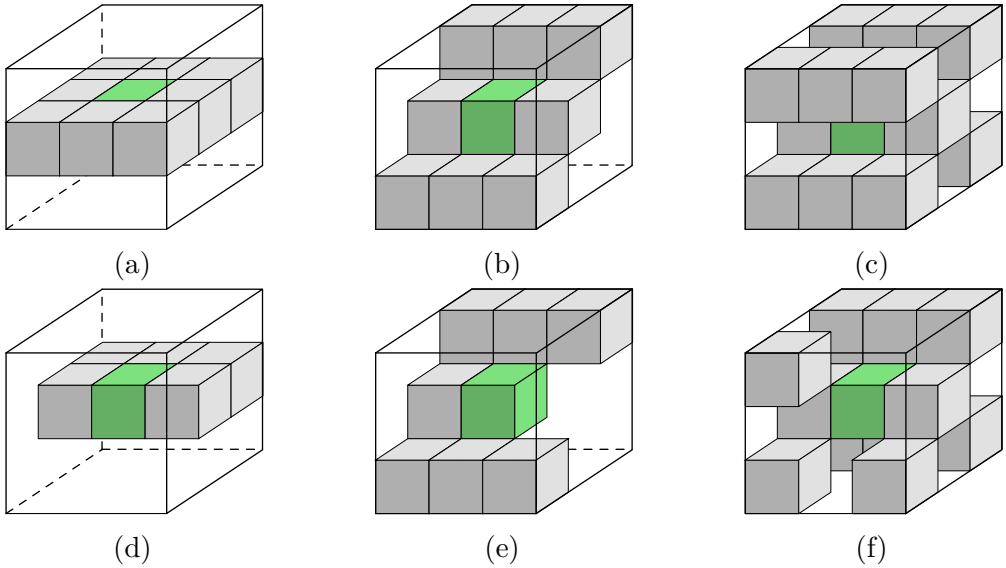


Fig. 3.14 Examples of surface points highlighted in green. If just using the configurations in Figure 3.14 as criteria for surface point, only the green voxels in (a), (b) and (c) can be detected as on the medial surface and surface points in (d), (e) and (f) are missed.

The above discussion leads to the definition: a solid voxel v is a surface point if and only if $\forall vert \in v$

$$\begin{aligned} &\text{configuration of } Oct(vert) \in \\ &\quad \{00001111, 00110011, 01010101, 11000011, 10011001, 10100101\} \quad \text{or} \end{aligned} \tag{3.14a}$$

$$n_3^{(vert)} < 3. \tag{3.14b}$$

where $Oct(vert)$ is the octant associated with vertex $vert$ and $n_3^{(vert)}$ is the number of hexahedrons overlapping $Oct(vert)$, see Section 3.1.1. Condition (3.14a) detects whether the voxel has octants lying on complete surfaces and (3.14b) covers the cases where some octants of the voxel have incomplete surfaces.

Replacing the 'end voxel' with the 'surface point' defined in (3.14), Algorithm 3.1 can yield surface skeletons instead of curve skeletons.

3.3 Skeletonisation examples

This section checks the correctness, robustness and efficiency of the skeletonisation algorithm 3.1 on various geometries from simple geometric primitives to non-trivial shapes.

3.3.1 Simple geometries

To start with, eight simple geometries are considered, of which four are with Euler characteristics of 1, two with 0 and last two with -1. In this section, χ is referred as the Euler characteristic of a volume mesh. The real medial axes and medial surfaces of these geometries are straightforward to determine, and to compare with the skeletonisation results.

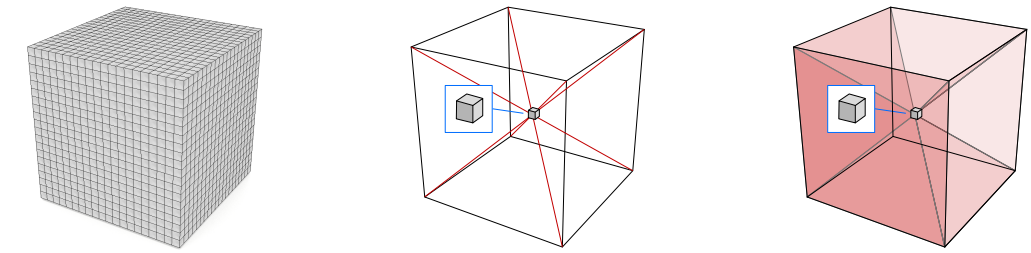
a. Volume mesh with $\chi = 1$

A cube on the grid of $20 \times 20 \times 20$ as shown in Figure 3.15a is skeletonised into curve skeleton using 11 steps shown in Figure 3.15b and surface skeleton using 11 steps in Figure 3.15c. To recap, there are six sub-steps in one removal step. Judging by (3.11), the Euler characteristics of the objects in Figures 3.15a, 3.15b and 3.15c are all equal to 1. So the homotopy of the thinning process is validated. Moreover, both the curve and surface skeleton are one-voxel thick. The real medial axis of a cube consists of the four diagonals of the cube, as displayed as red lines in Figure 3.15b, and the thinning outcome is the centre voxel, which lies on the medial axis. The medial surfaces of a cube consists of 12 triangles with one vertex at the centre and one edge at the cube edge, shown as the red transparent triangular planes in Figure 3.15c. The obtained surface skeleton (the centre voxel) is precisely on the medial surface.

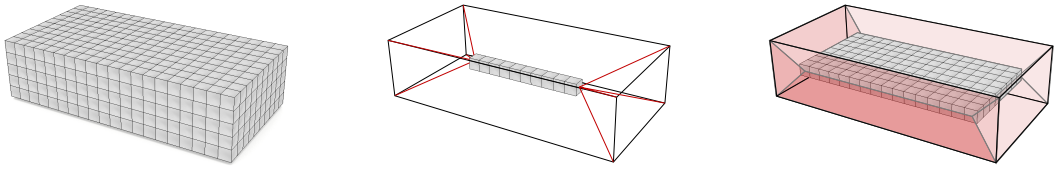
Another $\chi = 1$ volume mesh, a $20 \times 10 \times 5$-sized cuboid, is shown in Figure 3.15d. The real medial axis is shown as red lines in Figure 3.15e, where the curve skeleton with $\chi = 1$ is on. The Surface skeleton with $\chi = 1$ lies on the medial surface (red surfaces in Figure 3.15c) that consists of four triangles, eight trapeziums and one rectangular plane. To obtain the curve and surface skeleton, 4 and 3 removal steps are needed respectively.

The skeletonisation results of a sphere (on the grid of $40 \times 40 \times 40$) are shown in Figures 3.15h and 3.15i, and the results of an ellipsoid (on the grid of $80 \times 20 \times 20$) are in Figures 3.15k and 3.15l. These skeletons are homeomorphic to their before-removal counterparts in Figures 3.15g and 3.15j. The number of removal steps for skeletonising the sphere to curve and surface skeleton are 13 and 13 respectively, and for ellipsoid are 8 and 8 respectively.

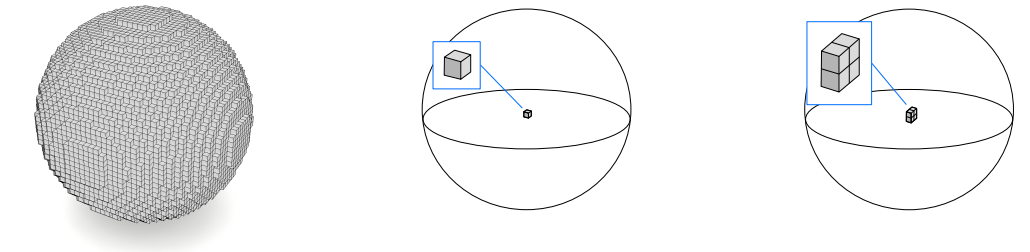
Note that the asymmetry of the skeletons are caused by the sequential layer-by-layer deletion used in the thinning algorithm. As stated in Section 3.2.2, such deletion may cause the resulting skeleton to be at most one-voxel-thick offset from the exact position [44], which is acceptable in most cases.



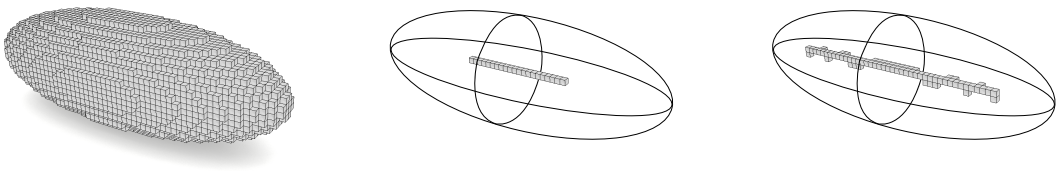
(a) Cube: $20 \times 20 \times 20$ (b) Curve skeleton of (a) (c) Surface skeleton of (a)



(d) Cuboid: $20 \times 10 \times 5$ (e) Curve skeleton of (d) (f) Surface skeleton of (d)



(g) Sphere: $40 \times 40 \times 40$ (h) Curve skeleton of (g) (i) Surface skeleton of (g)



(j) Ellipsoid: $80 \times 20 \times 20$ (k) Curve skeleton of (j) (l) Surface skeleton of (j)

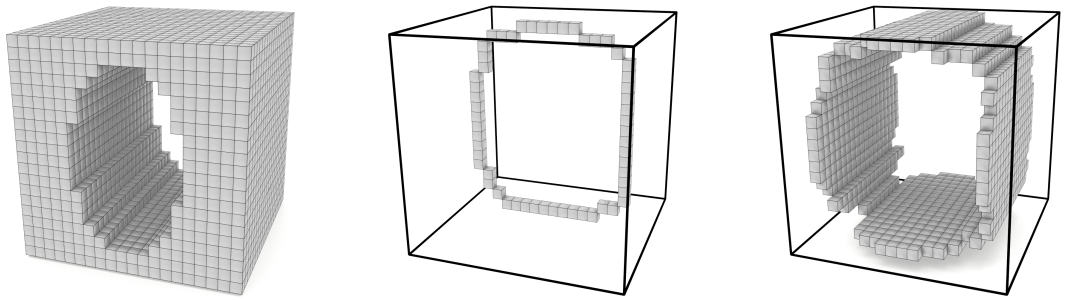
Fig. 3.15 Skeletonisation on simple geometries with $\chi = 1$. The black solid lines mark the shape of the object before the thinning. Red lines in (b) and (e) show the real medial axis, and red surfaces in (c) and (f) are real medial surfaces.

b. Volume mesh with $\chi = 0$

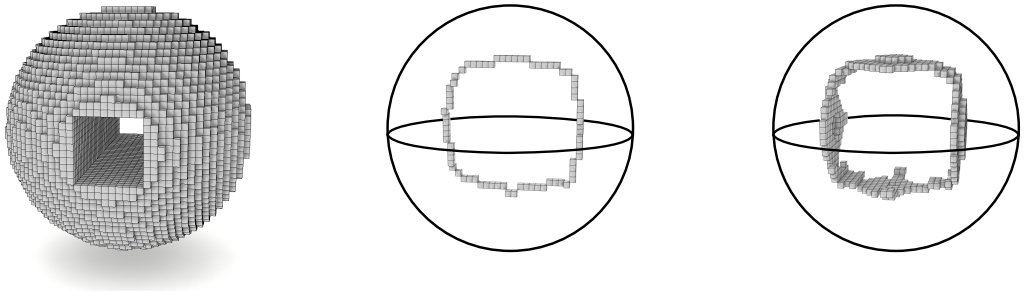
Two meshes with $\chi = 0$ are shown in Figures 3.16a and 3.16d. They are homeomorphic to a 1-torus.

The model in Figure 3.16a is a cube on the grid of $20 \times 20 \times 20$ subtracting an elliptic cylinder centred in the cube with radii of 8 and 5. Skeletonisation algorithm takes 6 removal steps to obtain the curve skeleton shown in Figure 3.16b and 4 steps for the surface skeleton shown in Figure 3.16c. Both the curve and surface skeleton preserve the topology of the original geometry, and they lie approximately in the middle between the elliptic cylindrical hole and the bounded faces of the cuboid.

A sphere on the grid of $40 \times 40 \times 40$ with a cuboid hole of size $10 \times 10 \times 40$ is shown in Figure 3.16d. The numbers of removal steps required for curve and surface skeletonisation are 8 and 7 respectively. The result skeletons are also with $\chi = 0$.



(a) Grid: $20 \times 20 \times 20$ (b) Curve skeleton of (a) (c) Surface skeleton of (a)



(d) Grid: $40 \times 40 \times 40$ (e) Curve skeleton of (d) (f) Surface skeleton of (d)

Fig. 3.16 Skeletonisation on simple geometries with $\chi = 0$. The black solid lines mark the shape of the object before the thinning.

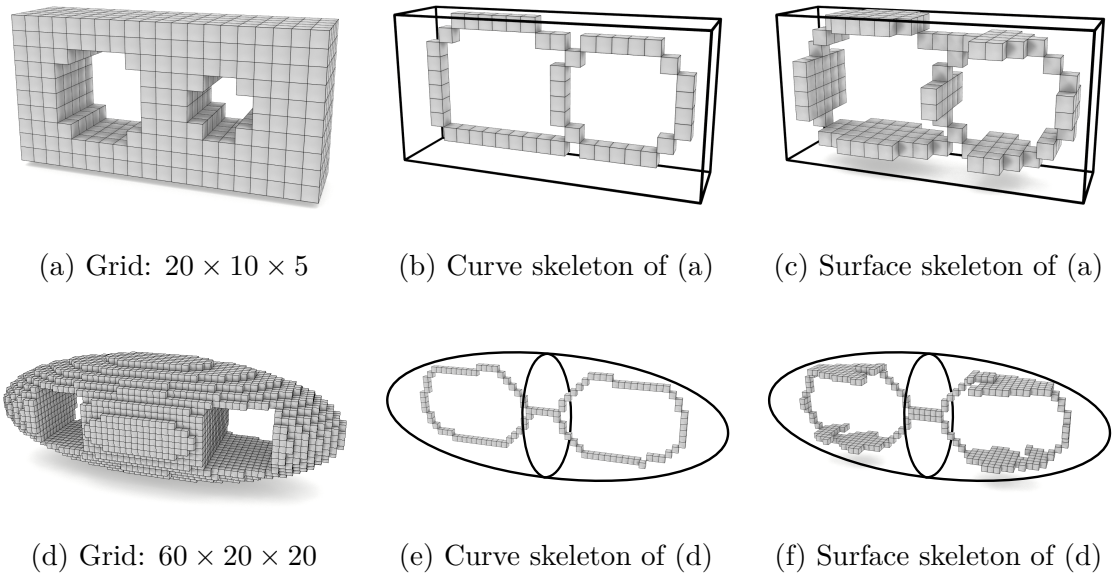
c. Volume mesh with $\chi = -1$

The volume meshes shown in Figures 3.17a and 3.17d are homeomorphic to a 2-torus, i.e. the object with two holes.

The object in Figure 3.17a is a $20 \times 10 \times 5$-sized cuboid with two cylindrical holes which have the circular cross-sections with radii of 3 and 2. 6 and 4 removal steps are required to obtain the curve skeleton shown in Figure 3.16e and the surface skeleton in

Figure 3.16f respectively. As is visually evident, both of the curve and surface skeleton have $\chi = -1$.

Another object shown in Figure 3.17d is an ellipsoid on the grid of $60 \times 20 \times 20$ subtracting two cuboids with cross-section of 10×10 along their longitudinal directions. For determining curve (see Figure 3.17e) and surface (see Figure 3.17f) skeleton, 8 and 7 removal steps are needed.



(a) Grid: $20 \times 10 \times 5$ (b) Curve skeleton of (a) (c) Surface skeleton of (a)

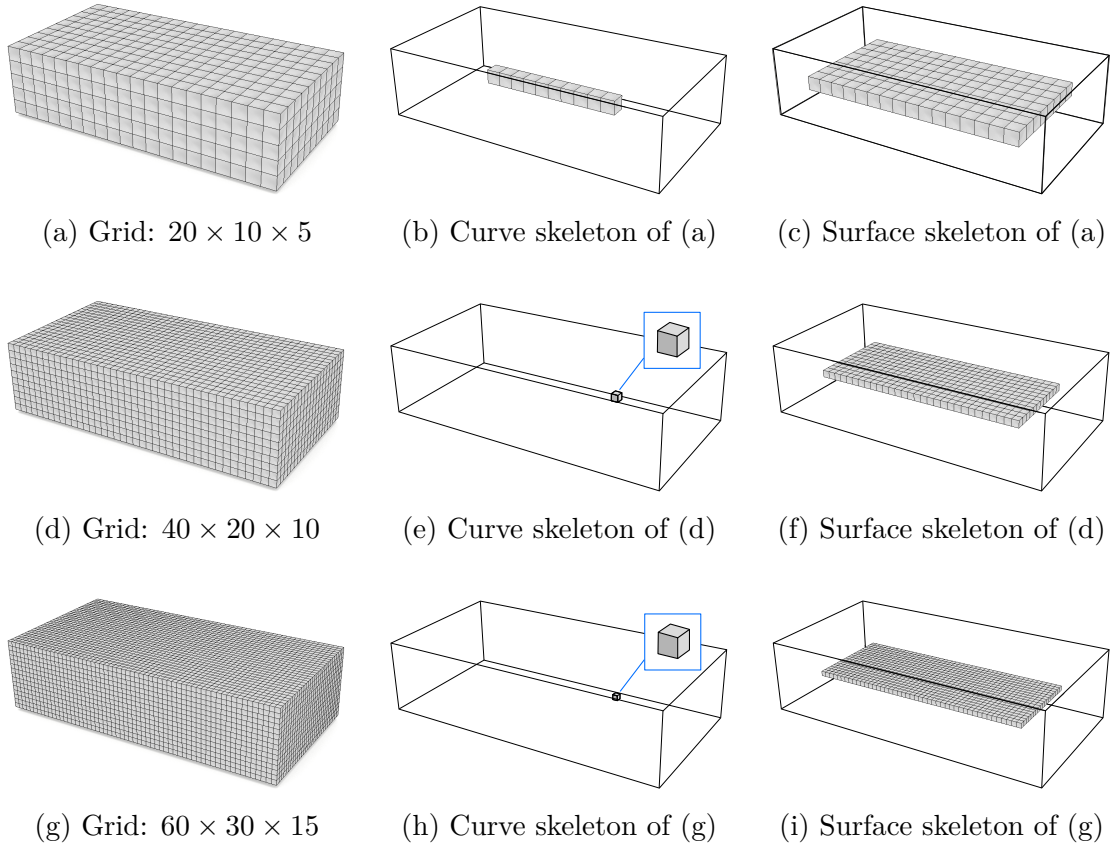
(d) Grid: $60 \times 20 \times 20$ (e) Curve skeleton of (d) (f) Surface skeleton of (d)

Fig. 3.17 Skeletonisation on simple geometries with $\chi = -1$. The black solid lines mark the shape of the object before the thinning.

3.3.2 Models with various resolutions

The results of Algorithm 3.1 on simple geometries are mesh-dependant. The models in Figures 3.18a, 3.18d and 3.18g are cuboids with the same length to width to height ratio of 4 to 2 to 1, while the grids for these three models are $20 \times 10 \times 5$, $40 \times 20 \times 10$ and $60 \times 30 \times 15$ respectively. As shown in Figure 3.18, the thinning algorithm on the same shape of three resolutions yields various results, especially in the case of curve skeletonisation process. In Figure 3.18b, the obtained curve skeleton is a voxel chain, while in Figures 3.18e and 3.18h, only one voxel is left, though these curve skeletons preserve the topology of their previous counterparts and lie close to the real medial axes. There are two reasons for the mesh-dependency: (i) The algorithm uses a sequence of symmetric layer removals, and the final results rely heavily on the number of layers of solid voxels along the deletion directions. (ii) If the shape is not

complex, the skeletonisation results can give voxels anywhere close to the real medial axes/surfaces. Once the geometry has a complex shape, the skeletonisation process yields much less mesh-dependant results. However, in most applications, the input geometry for skeletonisation is not as simple as those shown in Section 3.3.1.



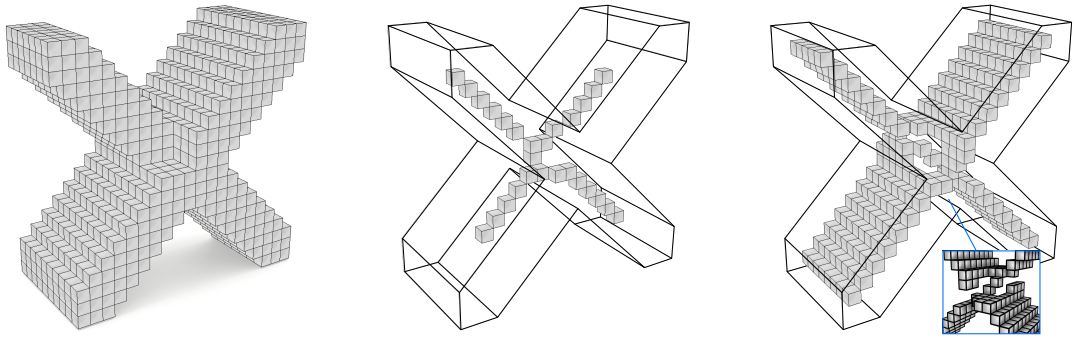
(a) Grid: $20 \times 10 \times 5$ (b) Curve skeleton of (a) (c) Surface skeleton of (a)

(d) Grid: $40 \times 20 \times 10$ (e) Curve skeleton of (d) (f) Surface skeleton of (d)

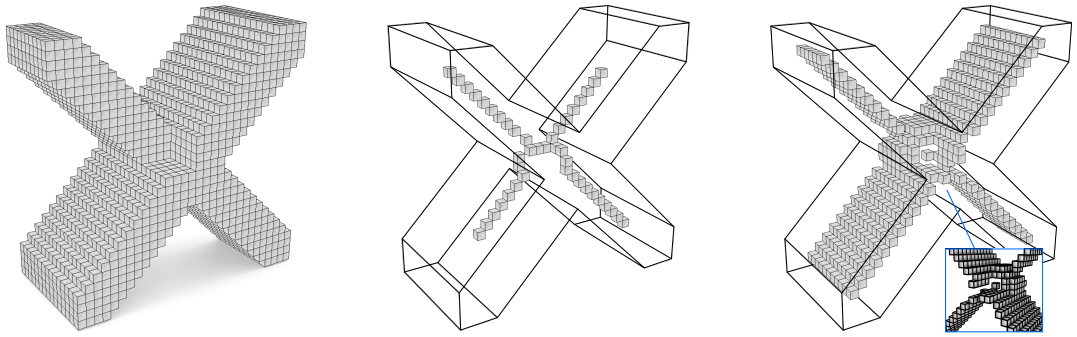
(g) Grid: $60 \times 30 \times 15$ (h) Curve skeleton of (g) (i) Surface skeleton of (g)

Fig. 3.18 Skeletonisation on Cuboid on three grid resolutions. The black solid lines mark the shape of the object before the thinning. Through curve skeletons in (b), (e) and (h) are different, the surface skeleton results are much less mesh-dependent, see (c), (f) and (i).

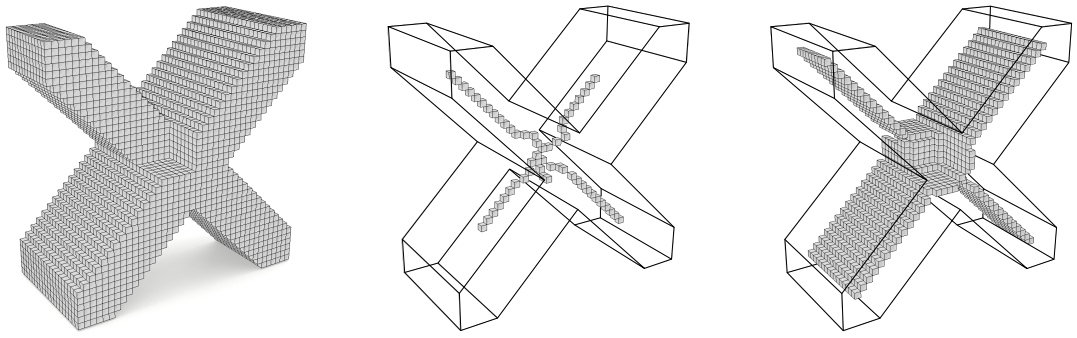
Figure 3.19 shows an X-shape geometry with a relatively more complex shape than a cuboid. This X-shape is modelled using three grids: $20 \times 10 \times 20$, $30 \times 15 \times 30$ and $40 \times 20 \times 40$, as shown in Figures 3.19a, 3.19d and 3.19g respectively. Despite of different grid resolutions, all the curve skeletons shown in Figures 3.19b, 3.19e and 3.19h give the X-shape voxel chains. As the resolution increases, the surface skeleton more closely resembles the real medial surface, see Figures 3.19c, 3.19f and 3.19i.



(a) Grid: $20 \times 10 \times 5$ (b) Curve skeleton of (a) (c) Surface skeleton of (a)



(d) Grid: $40 \times 20 \times 10$ (e) Curve skeleton of (d) (f) Surface skeleton of (d)



(g) Grid: $60 \times 30 \times 15$ (h) Curve skeleton of (g) (i) Surface skeleton of (g)

Fig. 3.19 Skeletonisation on X-shapes on three grid resolutions. The black solid lines mark the shape of the object before the thinning. The shapes of curve skeletons in (b), (e) and (h) are with slight difference. There is also not much difference between the shapes of surface skeletons, see (c), (f) and (i). Zoom-in windows in (c) and (f) are used to clarify there are no holes in the shape.

3.3.3 Horse geometry

Here Algorithm 3.1 is tested on a horse mesh model shown in Figure 3.20. The mesh has 1107 triangles. This example is used to show the mesh-independency of the thinning

algorithm on complicated shapes as well as how tagged voxels affect the skeletonisation process.

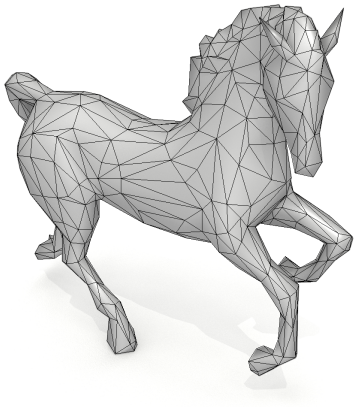


Fig. 3.20 The triangular mesh of the horse. The mesh consists of 1107 triangles and is obtained from GrabCAD [135].

a. Voxelisation

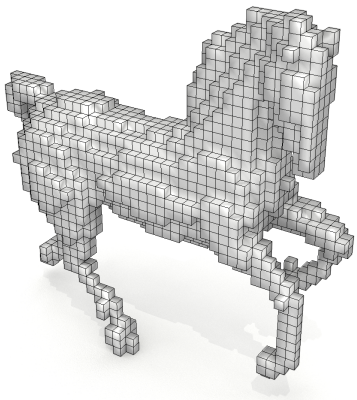
The conversion of a mesh into a binary image is as called in this thesis as *voxelisation*. First, openVDB [89] is used to compute the level-set function for the horse mesh, see Section 2.2.4, which is on a grid encompassing the size of the bounding box of the horse mesh. Then voxels with the given voxel size are generated to fill the entire grid. A voxel is identified as solid if the level set function value at its centroid is non-negative; otherwise, the voxel is set as void.

The four models voxelised using various grids are shown in Figures 3.21a, 3.21b, 3.21c and 3.21d with the grid of $42 \times 41 \times 16$, $83 \times 81 \times 32$, $124 \times 122 \times 48$ and $165 \times 162 \times 64$ respectively; details of voxelisations are in Table 3.1.

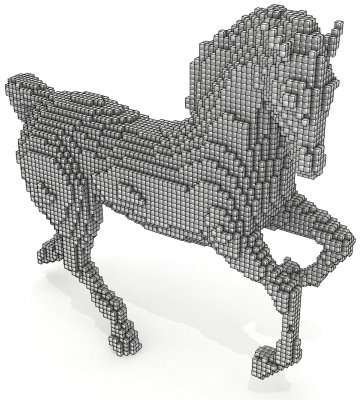
Table 3.1 Grid parameters used for voxelising the horse mesh

	Voxel grid #voxels	Voxel size	Voxel model #voxels
Grid 1	$42 \times 41 \times 16$	1	3237
Grid 2	$83 \times 81 \times 32$	0.5	25806
Grid 3	$124 \times 122 \times 48$	0.33	87152
Grid 4	$165 \times 162 \times 64$	0.25	206463

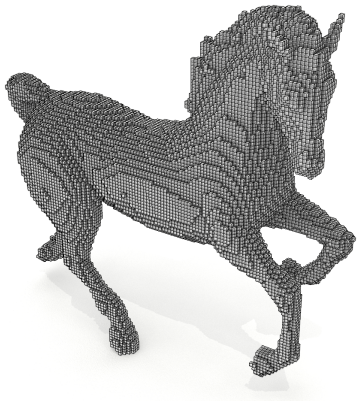
The voxel model on Grid 1 is very coarse with only 3237, which cannot capture the features such as the ear and head shape. By contrast, the model on Grid 4 with 206463



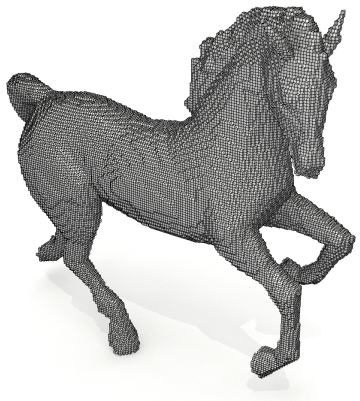
(a) Voxel model on Grid 1: $42 \times 41 \times 16$



(b) Voxel model on Grid 2: $83 \times 81 \times 32$



(c) Voxel model on Grid 3: $124 \times 122 \times 48$



(d) Voxel model on Grid 4: $165 \times 162 \times 64$

Fig. 3.21 Voxelised models of the horse mesh on four grid resolutions. Voxel model on Grid 1/Grid 2/Grid 3/Grid 4 has 3237/25806/87152/206463 voxels.

voxels is nearly 60 times finer than the model on Grid 1. Besides these two models, the models on Grid 2 and Grid 3 consist of 25806 and 87152 voxels respectively.

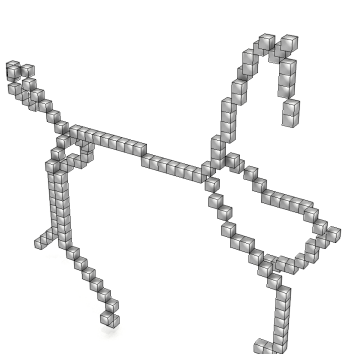
b. Curve skeletonisation

With the voxel models at hand, the curve skeletonisation algorithm is applied on the horse models. As shown in Figure 3.22, the curve skeletons contain much fewer voxels, but store the essential topological and geometric features of the horse. Just looking at the four curve skeletons in Figure 3.22, one can see the shape and the pose of the horse. It can also be visually confirmed that the skeletons, despite of resolutions, appear to have the same topology as their non-skeletonised counterparts.

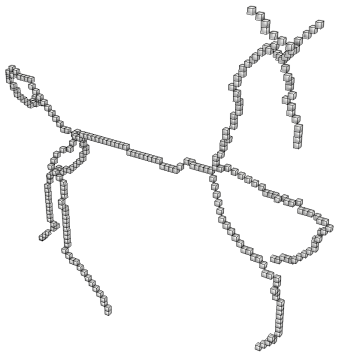
Table 3.2 Comparison of curve skeletonisations of the horse mesh on four grids

	Curve skeleton #voxels	Removal steps	Time per step	Time† in total
Grid 1	134	5	0.90 s	4.51 s
Grid 2	282	8	7.74 s	61.89 s
Grid 3	445	11	26.18 s	288.04 s
Grid 4	606	15	58.53 s	882.45 s

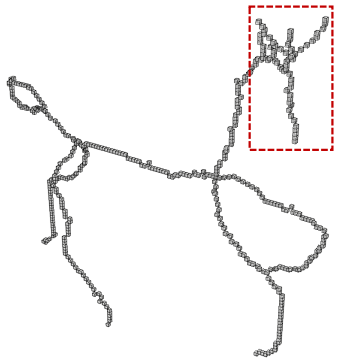
† The time cost used here is taken as the average time cost of three simulations. All simulations run on a MacOS machine with 6GB RAM and one thread of 2.5GHz Intel Core i7-4870HQ.



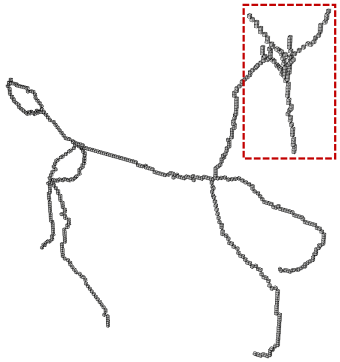
(a) Curve skeleton of the model on Grid 1



(b) Curve skeleton of the model on Grid 2



(c) Curve skeleton of the model on Grid 3



(d) Curve skeleton of the model on Grid 4

Fig. 3.22 Curve skeletons of the horse mesh on four grids. The curve skeleton of the model on Grid 1/Grid 2/Grid 3/Grid 4 contains 134/282/445/606 voxels. The parts highlighted in red windows show the geometric features have been captured by the high-resolution grids.

Table 3.2 records the result voxel number, removal steps and proceeding time of the skeletonisation process. As can be seen in Table 3.2, both the number of solid voxels in the voxel model and the number of removal steps increase when the grid becomes

finer. This is because there are more voxels covering the medial axis in the finer mesh. The number of skeleton voxels increases mainly because the smaller voxels can better capture the topology of the horse mesh. The reported extremely short runtimes include only skeletonisation (no implicitisation or voxelisation) and confirm the efficiency of the skeletonisation algorithm. The time consumed per step on these four models shows its linear relationship to the voxel number of the non-skeletonised model, which shall be discussed in detail in the next example.

Comparison of the skeletons on Grid 3 and Grid 4 in Figures 3.22c and 3.22d to those on Grid 1 and 2 in Figures 3.22a and 3.22b shows the former two contain more short branches. These short branches are highlighted in red windows in Figures 3.22c and 3.22d, which extend to the end of the horse mane and ears. In general, the skeleton of finer mesh can more easily capture the geometric features such as sharp edges and narrow corners, but it becomes more sensitive to boundary surface noise. The undesired short branches of curve skeleton can be pruned using the method introduced in Section 4.1.5.

c. Surface skeletonisation

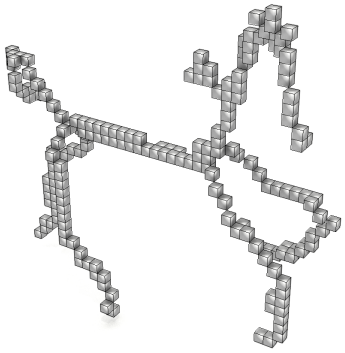
These four voxel-based horse models are also skeletonised into surface skeletons, and the parameters associated with the surface skeletonisation process are given in Table 3.3.

The resulting surface skeletons are shown in Figures 3.23a, 3.23b, 3.23c and 3.23d. The torso, legs and tail of the horse have the shapes similar to tubes, so large thin surfaces are expected to appear on the surface skeleton at these parts. However, horse manes are modelled in somewhat thin plates, see Figures 3.21 and 3.23, which lead to a relatively thin surface at the horse mane positions of the surface skeletons, as marked in the red windows in Figure 3.23.

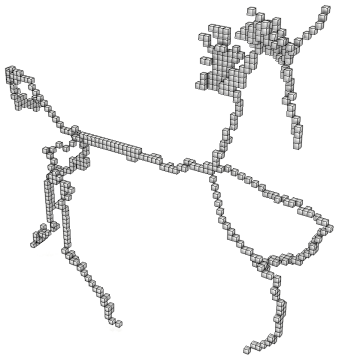
Table 3.3 Comparison of surface skeletonisations of the horse mesh on four grids

	Surface skeleton #voxels	Removal steps	Time per step	Time† in total
Grid 1	194	4	0.96 s	3.86 s
Grid 2	522	8	7.60 s	60.80 s
Grid 3	1031	11	27.67 s	304.42 s
Grid 4	1641	15	56.57 s	848.55 s

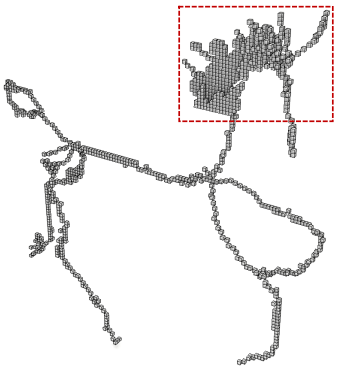
† The time cost used here is taken as the average time cost of three simulations. All simulations run on a MacOS machine with 6GB RAM and one thread of 2.5GHz Intel Core i7-4870HQ.



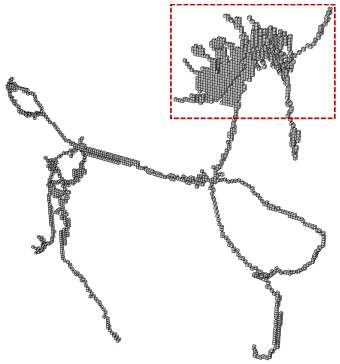
(a) Surface skeleton of the model on Grid 1



(b) Surface skeleton of the model on Grid 2



(c) Surface skeleton of the model on Grid 3



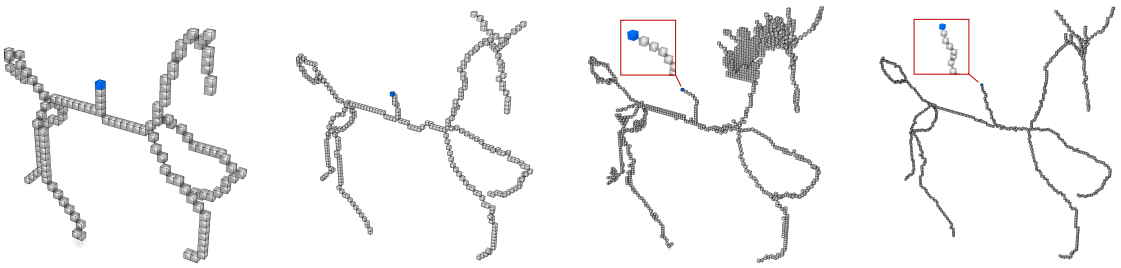
(d) Surface skeleton of the model on Grid 4

Fig. 3.23 Surface skeletons of the horse mesh on four grids. The surface skeleton of the model on Grid 1/Grid 2/Grid 3/Grid 4 contains 194/522/1031/1641 voxels. The parts highlighted in the dash red window show the thin surfaces represent the medial surface of the horse mane.

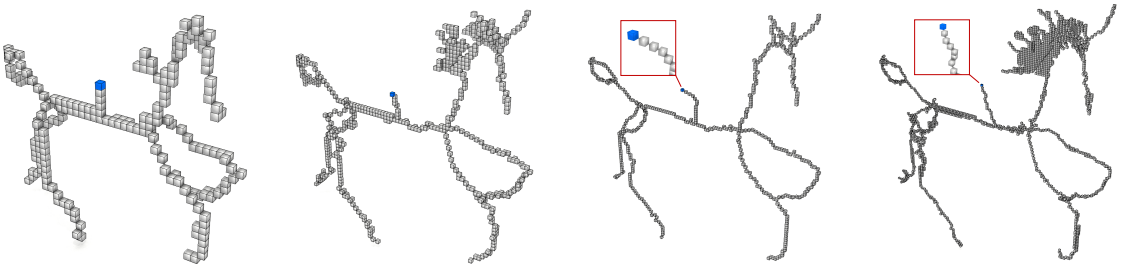
d. Skeletons of the model with tagged voxels

This part studies whether the introduction of irremovable tagged voxels will negatively affect the skeletonisation process. First, one voxel at the middle of the horse's back is tagged. Note that, the choice of tagged voxels is just for illustration. The resulting curve and surface skeletons on four grids are given in Figure 3.24. Comparison of Figure 3.24 to Figures 3.22 and 3.23 shows that the curve and surface skeletons of the model without tagged voxels extend to connect the tagged voxel. The resulting skeletons are still being one object. Next, voxels on the horse's bottom belly are tagged. As can be seen in Figure 3.25, the original curve and surface skeletons also extend to reach the centre of the tagged voxel group.

Because of the deletion directions in the skeletonisation process are given in a symmetric sequence, the voxel on such extending path from the original curve skeleton

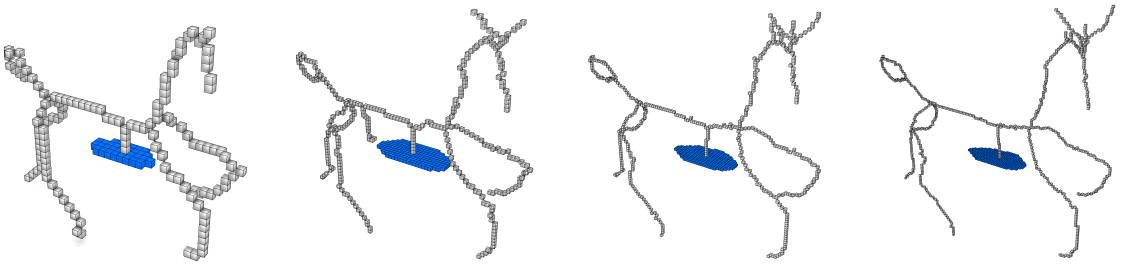


(a) Grid 1: Curve skeleton (b) Grid 2: Curve skeleton (c) Grid 3: Curve skeleton (d) Grid 4: Curve skeleton



(e) Grid 1: Surface skeleton (f) Grid 2: Surface skeleton (g) Grid 3: Surface skeleton (h) Grid 4: Surface skeleton

Fig. 3.24 Curve and surface skeletons of the horse mesh with one voxel tagged on four grids. The tagged voxel on the horseback is coloured in blue.



(a) Grid 1: Curve skeleton (b) Grid 2: Curve skeleton (c) Grid 3: Curve skeleton (d) Grid 4: Curve skeleton

(e) Grid 1: Surface skeleton (f) Grid 2: Surface skeleton (g) Grid 3: Surface skeleton (h) Grid 4: Surface skeleton

Fig. 3.25 Curve and surface skeletons of the horse mesh with voxel group tagged on four grids. The tagged voxels on the bottom belly are coloured in blue.

to the tagged one is guaranteed to have the approximate same distance to all closest boundary surfaces, as shown in Figures 3.24a to 3.24d, 3.25a to 3.25d. Such symmetry can also be observed in the generated surface skeletons in Figures 3.24e to 3.24h, 3.25e to 3.25h.

Therefore, the introduction of tagged voxels into skeletonisation process does not damage the homotopy of the skeletonisation process, and in the mean time, a connecting path between the original skeleton and the tagged voxels can be established. The computing time of thinning process on models with tagged voxels is also checked. It turns out that the process is not burdened by the tagged voxels.

3.3.4 Quadcopter geometry

A CAD model of a quadcopter is considered here to display the robustness and efficiency of the skeletonisation algorithm. The CAD quadcopter model shown in Figure 3.26a has a non-trivial topology and has been designed in a CAD system. The geometry of the quadcopter is approximated with the STL[1] mesh shown in Figure 3.26b. This STL mesh consists of 1086791 triangles having about 1000 times more triangles than the horse mesh.

(a) CAD model (b) STL mesh

Fig. 3.26 CAD model of the quadcopter. The quadcopter is designed in a CAD system as in (a). Its approximated STL mesh using 1086791 triangles is shown in (b). The quadcopter CAD model is obtained from open source CAD model library, GrabCAD [135].

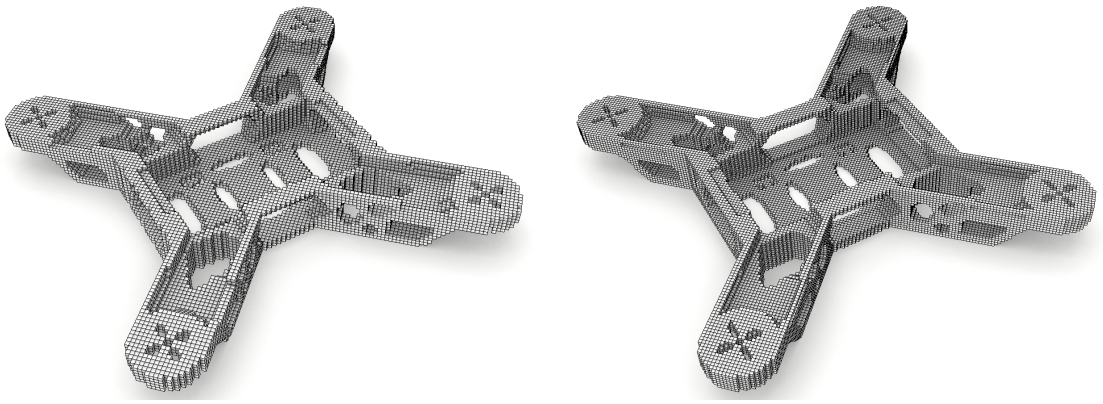
[1]STL file uses piece-wise triangulated surfaces to approximate the trimmed NUBRS surfaces on a CAD model.

a. Voxelisation

To investigate the efficiency and scaling of the introduced skeletonisation algorithms, four grids are deployed to extract the skeletons from the model. The voxelisation is conducted using the same method as used in the horse example. The voxel models of the quadcopter model in Figures 3.27a, 3.27b, 3.27c and 3.27d are obtained with the

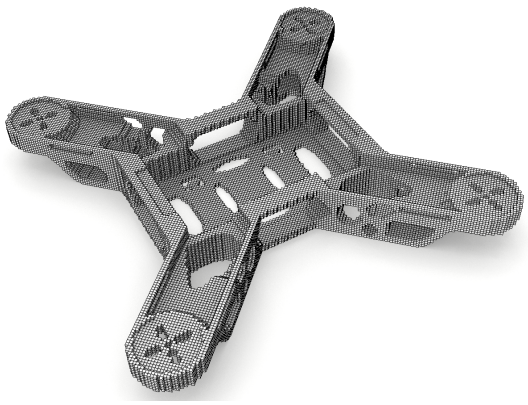
Table 3.4 Grid parameters used for Voxelising the quadcopter mesh

	Voxel grid #voxels	Voxel size	Voxel model #voxels
Grid 1	$142 \times 14 \times 142$	0.966	20142
Grid 2	$178 \times 18 \times 178$	0.770	39813
Grid 3	$208 \times 21 \times 208$	0.660	60320
Grid 4	$227 \times 23 \times 227$	0.604	80048

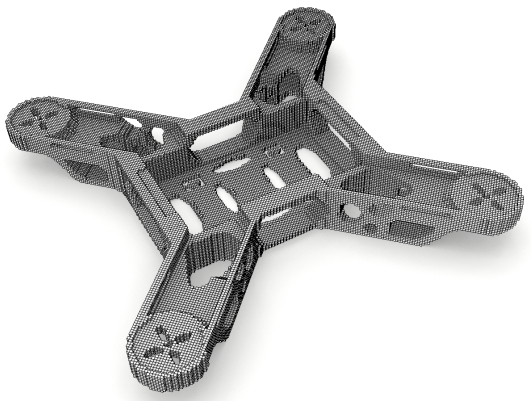


(a) Grid 1: $142 \times 14 \times 142$

(b) Grid 2: $178 \times 18 \times 178$



(c) Grid 3: $208 \times 21 \times 208$



(d) Grid 4: $227 \times 23 \times 227$

Fig. 3.27 Voxelised models of the quadcopter mesh on four grid resolutions. Voxel model on Grid 1/Grid 2/Grid 3/Grid 4 has 20142/39813/60320/80048 voxels.

grid size of 0.966, 0.770, 0.660 and 0.604 respectively, as given in Table 3.4. With the decrease in voxel size the grid becomes finer and the number of voxels in the model increases from 20142 via 39813 and 60320 to 80048.

Intentionally the Grid 2 is set to have double the number of voxels as on Grid 1, and the number of voxels on Grid 1 is tripled in Grid 3 and quadrupled in Grid 4; this is to highlight the linear time complexity of the thinning algorithm later.

b. Curve skeletonisation

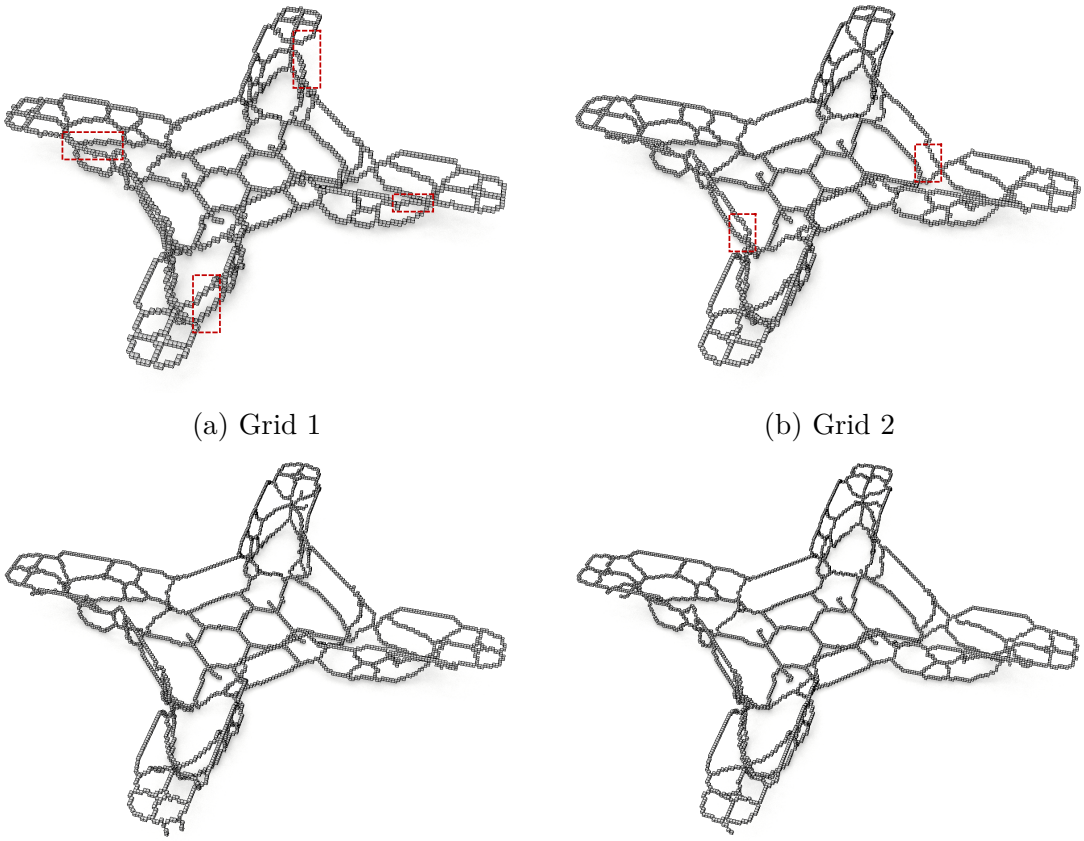
Skeletonisations of four voxelised models with the introduced thinning algorithm yield the voxel chain skeletons in Figures 3.28a, 3.28b, 3.28c and 3.28d respectively. It can be visually confirmed that the skeletons, despite of different resolutions, appear to have the same topology as the former voxel models in Figure 3.27, and for exact Euler characteristics of these skeletons, they are given at the end of this example.

To recap, the recorded short runtimes include only skeletonisation, and the efficiency of the skeletonisation algorithm is again confirmed. Moreover, note that the average time per removal step is approximately linear with respect to the number of the voxels in the voxel model, i.e. the number of voxels in Grid 2 doubles the number in Grid 1. The average time for one removal step in skeletonisation on Grid 2, 11.79s, is roughly two times longer than it on Grid 1, 6.00s. Similarly, the curve skeletonisation on Grid 3 and Grid 4 consume 17.22s and 21.93s per step, which are three times and four times than the time cost of each step on Grid 1. It is as expected that Algorithm 3.1 is with linear complexity [44].

Table 3.5 Comparison of curve skeletonisations of the quadcopter mesh on four grids

	Curve skeleton # voxels	Removal steps	Time per step	Time† in total
Grid 1	1322	5	6.00 s	29.98 s
Grid 2	1770	5	11.79 s	58.96 s
Grid 3	2135	6	17.22 s	103.30 s
Grid 4	2364	6	21.93 s	131.58 s

† The time cost used here is taken as the average time cost of three simulations. All simulations run on a MacOS machine with 6GB RAM and one thread of 2.5GHz Intel Core i7-4870HQ.



(a) Grid 1 (b) Grid 2

(c) Grid 3 (d) Grid 4

Fig. 3.28 Curve skeletons of the quadcopter mesh on four grids. The curve skeletonisation of 20142-/39813-/60320-/80048-voxels model on Grid 1/Grid 2/Grid 3/Grid 4 takes 5/5/6/6 steps to thin a model into a 1322-/1770-/2135-/2364-voxels curve skeleton. Red windows highlight missing parts of the curve skeletons in (a) and (b) compared to finer grids in (c) and (d).

c. Surface skeletonisation

The medial surfaces are extracted as in Figure 3.29. The well-preserved topologies during the medial surface skeletonisations are validated with the Euler characteristics given at the end of this example. With the condition stated as (3.14), these voxel models have been skeletonised into one-voxel thick surfaces.

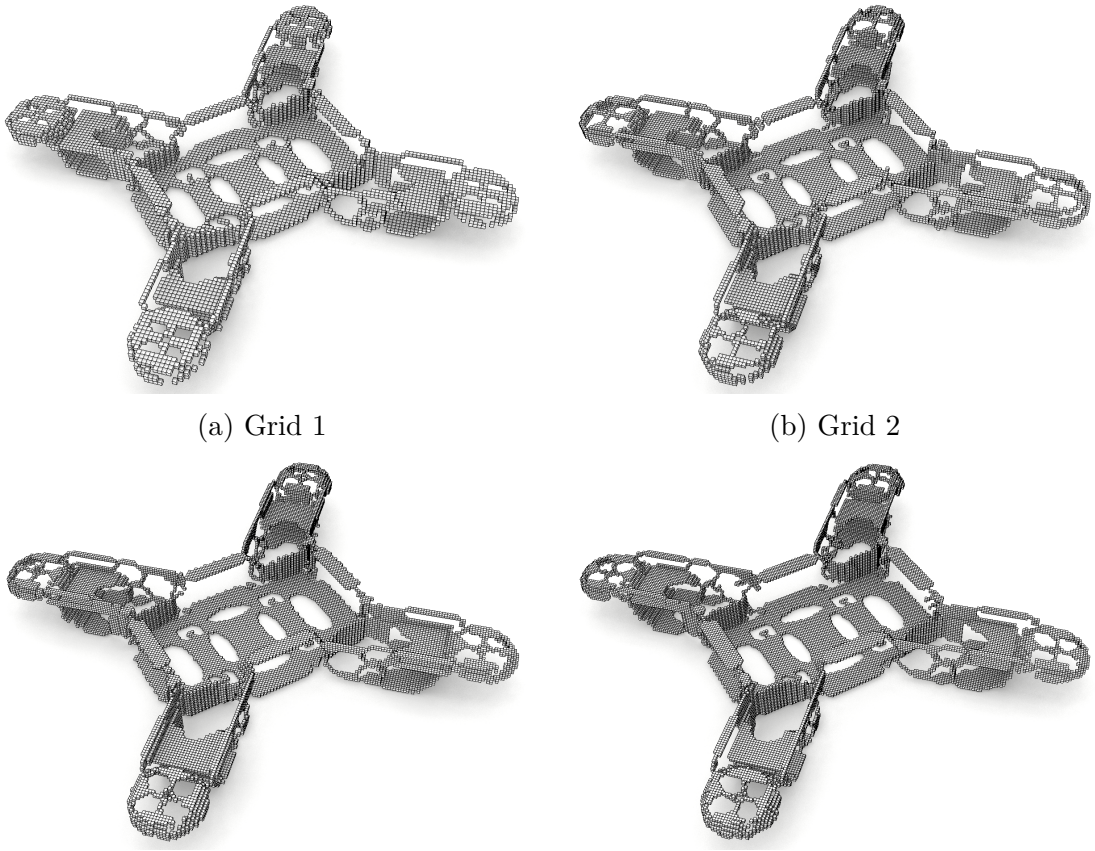
The average time per removal step also proves to be closely linearly related to the number of voxels in the original model. For example, time per step in the models on Grid 2, Grid 3 and Grid 4 are 11.32s, 15.66s and 21.95s respectively, which approximately doubles, triples and quadruples the time per one step in skeletonising the model on Grid 1. For a thin shape like this quadcopter one can barely see the difference in total

time consumption between curve skeletonisation and surface skeletonisation, while for

Table 3.6 Comparison of surface skeletonisations of the quadcopter mesh on four grids

	Surface skeleton # voxels	Removal steps	Time per step	Time† in total
Grid 1	5413	4	6.51 s	26.82 s
Grid 2	8960	5	11.32 s	56.60 s
Grid 3	12413	6	15.66 s	93.94 s
Grid 4	14416	6	21.95 s	131.72 s

† The time cost used here is taken as the average time cost of three simulations. All simulations run on a MacOS machine with 6GB RAM and one thread of 2.5GHz Intel Core i7-4870HQ.



(a) Grid 1 (b) Grid 2

(c) Grid 3 (d) Grid 4

Fig. 3.29 Surface skeletons of the quadcopter mesh on four grids. The curve skeletonisation of 20142-/39813-/60320-/80048-voxels model on Grid 1/Grid 2/Grid 3/Grid 4 takes 4/5/6/6 steps to thin a model into a 5413-/8960-/12413-/14416-voxels surface skeleton.

thicker shapes, larger difference can be expected, and surface skeletonisation normally uses fewer steps than the curve skeletonisation.

Note that the quadcopter is designed normally using shell structures. In this case, the surface skeleton represents the model more intuitively than the curve skeleton. The surface skeleton of the quadcopter can be visually segmented into shell/plate parts and the frame-like parts. Then designers can gain understandings of the mechanical behaviour of each segment, i.e. it performs more like a shell or a beam. This can be useful in assigning semantics to the segments of this quadcopter, which can be a part of the future work for this thesis, see Section 6.2.

d. Euler characteristics of the objects during the thinning process

Different from the previous examples, the Euler characteristics of the quadcopter model or its skeletons cannot be easily determined by eyes. Accordingly relation (3.1) is used to compute the Euler characteristics. All Euler characteristics and corresponding topological entities of the mesh are listed in Table 3.7.

As validated by the Euler characteristics listed in Table 3.7, both the curve and surface skeletonisation preserve the topology of the quadcopter mesh, i.e. Euler characteristics are identical for the voxel model, skeletons on the same grid. The

Table 3.7 Topological entities and Euler characteristics of models on different grids

		#vertex n_0	#edge n_1	#face n_2	#volume n_3	χ
Grid 1	Model	33517	86292	72852	20142	−65
	Curve†	6639	12658	7276	1322	−65
	Surface‡	15641	34814	24521	5413	−65
Grid 2	Model	60871	160389	139264	39813	−67
	Curve	8797	16791	9697	1770	−67
	Surface	25050	56530	40373	8960	−67
Grid 3	Model	89060	237010	208201	60320	−69
	Curve	10802	20520	11784	2135	−69
	Surface	33664	76709	55389	12413	−69
Grid 4	Model	114324	307073	272728	80048	−69
	Curve	11934	22649	13010	2364	−69
	Surface	38743	88498	64102	14416	−69

† Curve skeleton.
‡ Surface skeleton.

difference in Euler characteristics between different grids is due to the different accuracy of the voxelisation process. With the voxel model on Grid 1, four slim horizontal links are missing since the precision of the grid is not sufficient to capture them. This leads to the missing horizontal members on the corresponding skeleton, as shown in the red windows in Figure 3.28a. Thus the skeletons on Grid 1 have four fewer holes than the skeletons on the Grid 3 and Grid 4, which causes the difference of 4 in their Euler characteristics. The voxelisation on Grid 2 misses two vertical members of the quadcopter mesh, which occur in the place highlighted in the red window in Figure 3.28b. As a result, the Euler characteristic of the model on Grid 2 is 2 more than those on Grid 3 and Grid 4, since models on Grid 3 and Grid 4 have 2 more holes.

Careful counting the holes on the quadcopter mesh gives 10 holes on the bottom plate, 4 holes on the side plate connected to the bottom plate, and 14 holes on each wings of the four. so in total $10 + 4 + 14 \times 4 = 70$ holes. The Euler characteristics of quadcopter model is $\chi = 1 - 70 = -69$ using relation (3.11). Therefore, the resolution of Grid 3 with size of 0.660 are already precise enough to capture all topological features of the quadcopte model.

CHAPTER 4

Frame extraction and CAD model generation

This chapter presents the method based on graph algorithms to process topology-preserved skeleton models into structural frame models then reconstruct the frame to compact CAD model. To represent the voxel chain obtained through skeletonisation, using nodes and edges rather than voxels can give not only a simpler geometric representation but also better understanding of structural semantics of the chain branches. Hence this chapter starts with the conversion of a topology-preserved voxel chain to a frame model in Section 4.1 which begins with an introduction to graph models. The conversion of voxel chain models to graph models is based on the principles stated in Sections 4.1.2 for graph nodes and 4.1.3 for graph edges. Useful tools are also provided to process graph models to produce more structurally sound frames in Section 4.1.4 and 4.1.5. Next, Section 4.2 elaborates the CAD model regeneration from graph models. The output models can either be subdivision surfaces as introduced in Section 4.2.1 or B-rep models in Section 4.2.2. Finally, Section 4.3 gives an example to illustrate the whole process to reconstruct a voxel model into a compact CAD model.

4.1 Frame extraction

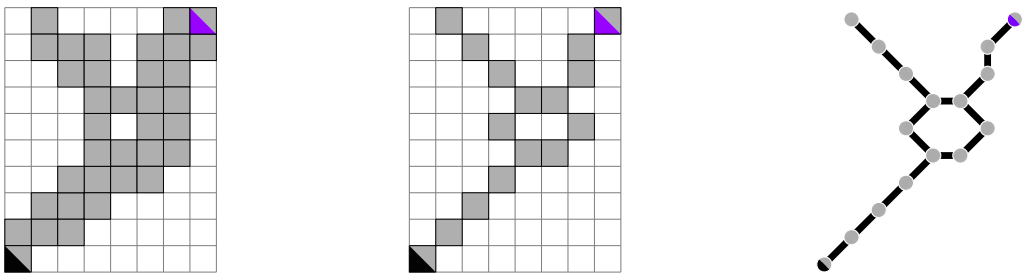
The skeleton provides both the connectivity and the geometry of the frame. Although it is feasible to obtain the member cross-sections from the voxel model, in the implementation used in this thesis, cross-sections are obtained from size optimisation of the frame, as will be discussed in Section 5.1. In conversion of the voxel chain skeleton to a frame model, Graph model is used to represent the frame model.

Graph is a well-proven tool used in mathematics, computer science and operations research, which in particular is developed to treat problems having network characteristics such as frame connectivities [136]. Though similar idea of representing frames using graphs can be found in [137, 138], currently there is no research giving a clear guidance of converting voxel chains into graph models. Hence this section aims at providing algorithms and tools to enable such conversion. Here, the curve skeleton is treated as a weighted undirected graph to make use of its incidence matrix. The frame model provides a very compact representation of the optimised structure, which can be used to generate a subdivision surface model, or a parametric solid model using the scripting interface or *Application Programming Interface* (API) of a CAD system.

4.1.1 Graph model

Before getting into details, a simple and direct way to convert a one-voxel thick curve skeleton into a graph is discussed first. The graph consists of *nodes* and *edges*, denoted as $\mathcal{G} = (Sk(\mathcal{M}), \mathcal{E})$. Here $\mathcal{M}$ is the topologically optimised hexahedral mesh, $Sk(\mathcal{M})$ is a set of voxels on the curve skeleton of $\mathcal{M}$. For simplicity, skeleton is referred to as curve skeleton in this chapter. The edges $\mathcal{E}$ are 2-element subsets of $\mathcal{M}$. One edge is associated with two voxels in a 26-neighbourhood. Each of the nodes has an associated coordinate which is initially equal to the centroid of the corresponding voxel.

For example, the voxel model in Figure 4.1a is skeletonised using Algorithm 3.1 into a voxel chain shown in Figure 4.1b. It is assumed that there are two tags used in the voxel model, and the tagged voxels are coloured half in black for tag_1 and purple for tag_2. For structural and mechanical engineering applications, tags are used to represent the locations of loadings and boundary conditions. Then the voxel chain is



(a) Voxel model (b) Voxel chain from (a) (c) Graph model from (b)

Fig. 4.1 An example of a simple way of converting a voxel chain to a graph model. Voxel model (a) is skeletonised into skeleton (b). Graph (c) is generated by connecting the edges between 26-neighboured voxels (b).

expressed as the graph model in Figure 4.1c. This graph model contains 16 voxels and 16 edges. It is predictable that the number of voxels and edges will significantly increase when the model has greater complexity. Therefore, directly converting the edges in Figure 4.1c into frame members is neither feasible nor practical, since these short beams are ineffective in transferring loads as well as burden the frame optimisation. As a result a more compact graph is needed, which consists of only 'important' voxels instead of all voxels and the edges linking these 'important' voxels. Hence the types of voxels are required to be identified to locate these 'important' voxels.

4.1.2 Type classification

In order to identify the 'important' voxels mentioned in the previous section, the voxel and node types are defined based on their topological and user-defined features.

a. Voxel types

Topologically, the voxels on a skeleton are classified into three types of voxels [44]:

- A regular voxel has two 26-neighboured voxels.
- An end voxel has only one 26-neighboured voxels.
- A joint voxel has more than two 26-neighboured voxels.

Here the definition of an end voxel is the same as in Algorithm 3.1. Both end voxels and joint voxels represent topological and geometric features of the skeleton model [44], which are the 'important' voxels to be sought. In addition to preserving the topology,

(a) Voxels types are identified (b) Node types are identified

Fig. 4.2 Voxel and node type identification. (a) shows the type classification for voxels in a voxel chain. (b) shows the type classification for nodes in a graph. The end, regular, joint voxels/nodes are in blue, grey and green respectively, the tagged voxels/nodes are half coloured in black or purple.

the tagged voxels in the skeletonisation process are also kept as graph nodes for preserving user-defined features. As illustrated in Figure 4.2a, all voxels are classified as regular (grey), end (blue) or joint voxels (green). The voxels half coloured with black/purple are the tagged voxels (assuming there are two tags applied).

Through voxel type classification of the skeleton model, a more compact node set $\mathcal{V} \subset Sk(\mathcal{M})$ is found, which is a set of *featured voxels* (end, joint and tagged voxels).

b. Node types

The operations on graph models may change the connectivity of voxels. In order to maintain the graph model in its accurate form, the node types in the graph require subsequent rechecks. Similarly, the node types in graphs can be defined as:

- A regular node is shared by two edges.
- An end node is connected to only one edge.
- A joint voxel is shared by more than two nodes.

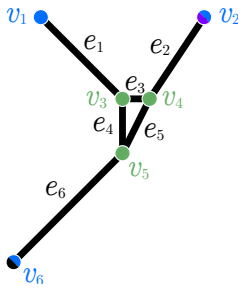
For example, Figure 4.2b shows the node type identification of a graph model, using similar colouring fashion as in Figure 4.2a.

4.1.3 Graph construction from voxel chain

Here, an edge consists of two voxels in $\mathcal{V}$, and these two voxels are connected through a path consisting of untagged regular voxels. The number of voxels between the beginning voxel and the ending voxel (including these two voxels themselves) of one edge is also recorded as the edge *weight*. This weighted undirected graph can be described using an *incidence matrix*: its rows correspond to the nodes of the graph and the columns to the edges. Each edge has two identical non-zero entries corresponding to the weight of the edge in voxel units[1].

Once all voxel types are identified, starting from the joint voxels, the voxel chains and their lengths are determined by marching along their 26-neighbours until a featured voxel is reached. During this process, the duplicate edges are ignored, i.e. edges with the same nodes. The obtained graph model is stored associated with an incidence matrix, as illustrated in Figures 4.3a and 4.3b. The corresponding algorithm is described in Algorithm 4.1.

[1]Alternatively, the weight of an edge can also be the Euclidean distance between two edge nodes. When using Euclidean distances as the weights, after every operation that changes node positions or edge connections, weights require to be updated by calculating the distance between new nodes.



(a) Graph

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	4	0	0	0	0	0
v_2	0	4	0	0	0	0
v_3	4	0	2	3	0	0
v_4	0	4	2	0	4	0
v_5	0	0	0	3	4	5
v_6	0	0	0	0	0	5

(b) Incidence matrix

Fig. 4.3 Graph extracted from the voxel chain in Figure 4.2a

Algorithm 4.1: Graph construction from the skeleton

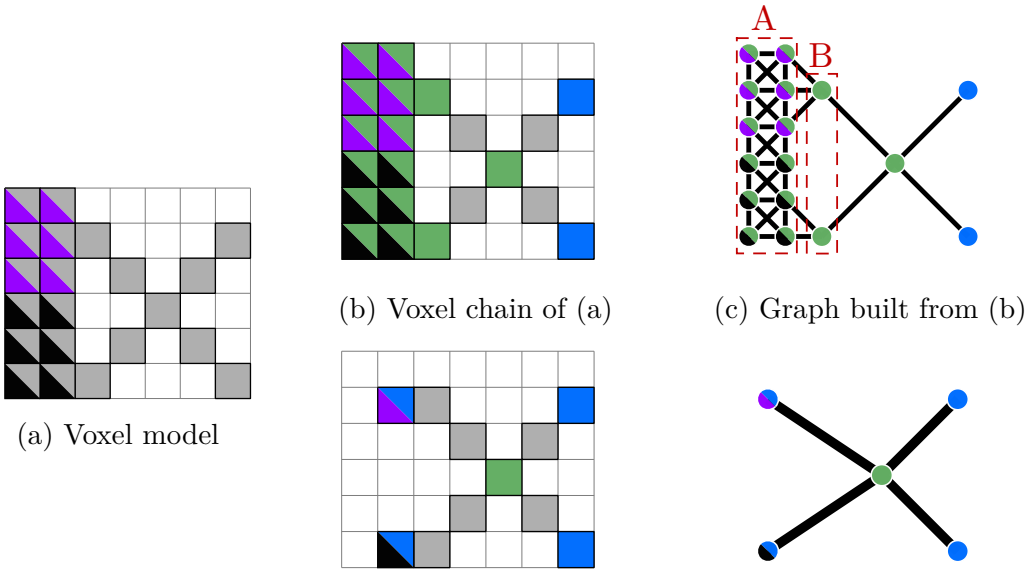
input : $\mathcal{V} \subset Sk(\mathcal{M})$, set of featured voxels[†]
$\mathcal{T} = \{tag_1, tag_2, ...\}$, the tags defined by the user
output : $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, graph generated from $Sk(\mathcal{M})$
$\boldsymbol{B}(\mathcal{G})$, incidence matrix of graph $\mathcal{G}$

$\mathcal{E}$: list of edges
$\mathcal{P}$: list of visited regular voxels
$\mathcal{W}$: list of weights
forall *voxel* $v \in \mathcal{V}$ **do**
  if v *is joint voxel* **then**
    forall *voxel* $s \in \mathcal{N}_{26}(v)$ **do**
      $w \leftarrow 1$ ▷ initialise the length of the edge as 1
      while s *is untagged regular voxel* **and** $s \notin \mathcal{P}$ **do**
        insert s into $\mathcal{P}$ ▷ mark s as visited
        t: the only unvisited neighboured voxel to s
        $s \leftarrow t$ ▷ procedure keeps marching along the regular voxels
        $w \leftarrow w + 1$
      if $\{v, s\} \notin \mathcal{E}$ **then**
        insert edge $\{v, s\}$ into $\mathcal{E}$ ▷ only the unique edge is stored
        $w \leftarrow w + 1$
        insert w into $\mathcal{W}$
$\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E})$ ▷ construct the graph
$\boldsymbol{B}(\mathcal{G}) \in \mathbb{R}^{\#\text{node} \times \#\text{edge}}$: initialise incidence matrix
forall *edge* $e_i = \{v_j, v_k\} \in \mathcal{E}$ **do**
  $\boldsymbol{B}(\mathcal{G})_{v_j e_i} \leftarrow w_i$ ▷ edge weight w_i is assigned to row v_j column e_i in $\boldsymbol{B}(\mathcal{G})$
  $\boldsymbol{B}(\mathcal{G})_{v_k e_i} \leftarrow w_i$ ▷ edge weight w_i is assigned to row v_k column e_i in $\boldsymbol{B}(\mathcal{G})$
return $(\mathcal{G})$, $\boldsymbol{B}(\mathcal{G})$

[†] Featured voxels are referred to the ones which are not untagged regular voxels, see Section 4.1.2.

Since all tagged voxels are kept during the skeletonisation process, clusters of tagged voxels remain. These clusters occur, for example, when the structure has loadings or boundary conditions applied on areas. In the case of determining a structural frame from a graph model, these clusters confuse Algorithm 4.1 and cause two issues: (i) These tagged voxels in clusters will be identified as joint voxels if they have more than two neighbours, and enormous short edges will be constructed linking these tagged voxels, see the region A in Figure 4.4c. These short edges come from clusters of tagged voxels gathered in the left part of the voxel chain in Figure 4.4a with their types identified in 4.4b. (ii) Any regular voxel neighboured to these tagged voxel clusters may be incorrectly identified as a joint voxel, e.g. voxels in region B in Figure 4.4c. These lead to a graph model with unnecessarily many nodes and edges, as shown in Figure 4.4c.

In order to eliminate the confusion caused by these clusters of tagged voxels, the *representative tagged voxels* are used. The positions of representative tagged voxels mark where the untagged voxel chains touch the tagged voxel clusters. The tagged voxels which are not identified as representative are regarded as redundant. Algorithm 4.2 are



(a) Voxel model

(b) Voxel chain of (a)

(c) Graph built from (b)

(d) Simplified (b)

(e) Graph built from (e)

Fig. 4.4 Clusters of tagged voxels may confuse the graph construction algorithm. Joint, end and regular voxels/nodes are coloured in green, blue and grey respectively. Black and purple are used for marking tagged voxels/nodes. Voxel chain (b) has two large clusters of tagged joint voxels, so graph (c) generates many short edges between these joints. Algorithm 4.2 simplifies (b) into (d) that leads to a desired graph model (e).

Algorithm 4.2: Identification of representative tagged voxels

input : $Sk(\mathcal{M})$, the voxel chain containing redundant tagged voxels
$\mathcal{T} = \{tag_1, tag_2, ...\}$, the tags defined by the user
update : $Sk(\mathcal{M})$: the simplified skeleton without redundant tagged voxels

```
1  forall tag_i ∈ T do
2      P: list of visited voxels with tag_i
3      R: list of representative tagged voxels of tag_i
4      forall v with tag_i do
5          while ∃s, (s ∈ N_26(v) and s ∉ P and s is not with tag_i) do
6              insert s into P        ▷ mark the neighboured untagged-as-tag_i voxel as visited
7              r ← argmin(dist†(s,t), t ∈ N_26(s) and t with tag_i)
8              insert r into R        ▷ the one with minimum distance becomes representative
9      forall v with tag_i do
10         if v ∉ P then
11             delete v from Sk(M)    ▷ remove redundant tagged voxels from skeleton
```

† $\text{dist}(s,t)$ is Euclidean distance between the centroids of the two voxels: s and t

given below to detect the representative tagged voxels and to delete the redundant ones. The outer loop of Algorithm 4.2 successively checks tags. Its inner loop is over voxels with the same tag, for example, tag_i. Once the algorithm locates a tagged-as-tag_i voxel v with the untagged-as-tag_i neighboured voxel s, it turns to search tagged-as-tag_i neighboured voxels of s. Then the voxel, tagged as tag_i, with the shortest distance among other tagged-as-tag_i voxels in $\mathcal{N}_{26}(s)$ is identified as the representative tagged voxel of tag_i. When all voxels with tag_i have been visited, the non-representative tagged voxels are deleted from the skeleton $Sk(\mathcal{M})$ and then move to the next tag until all tags are processed. This process ensures that every voxel is neighboured to at most one voxel with one certain tag that differs from the tag of the voxel being processed. Algorithm 4.2 can be used as a pre-process to Algorithm 4.1 if needed.

Applying Algorithm 4.2 on the skeleton in Figure 4.4b to simplify the skeleton into the voxel chain shown in Figure 4.4d. Then this voxel chain model can be converted using Algorithm 4.1 into a compact graph model in Figure 4.4e. One can directly use the graph model as a frame model by assigning cross-sectional parameters to the edges and applying the loadings and boundary conditions to tagged nodes. However, the frame model from a voxel chain possibly contains short members or zero-stress beams. In order to post-process the graph model, useful optional operations on incidence matrices are discussed in the following sections; the user can make use of them as one sees fit.

4.1.4 Edge collapse

In general, a skeletonisation result may have clusters of joint voxels. The joint voxels in close proximity to each other are connected by very short edges. It is not expedient to deduce from every joint voxel a graph node for the frame model because this will lead to acutely short members and impractical joint designs. In order to remove these short edges caused by clustered joint voxels without damaging the load path, edge collapse is used to merge graph nodes connected by short edges. Merging can be achieved by performing row and column removals on the incidence matrix of the graph, and short edges can be detected if they have weights of 2 (only two voxels in the edge). The edge collapse is divided into three distinct cases as follows:

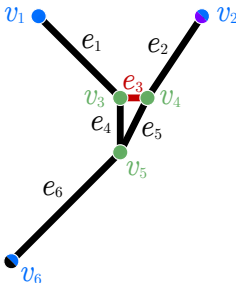
- If both nodes are tagged, edge collapse is not allowed.
- If only one node is tagged, the edge can be deleted, and the remaining node keeps the coordinate and the tag of the tagged one.
- If no node is tagged, the edge can be deleted, and the coordinate of the remaining node uses the average of coordinates of the two nodes.

The graph model in Figure 4.3a is used to illustrate the edge collapse. In Figure 4.5, the edge e_3 coloured as red is detected as a short edge because of $w_3 = 2$. The edge e_3 is collapsed by merging row v_4 to row v_3 and removing column e_3 in the incidence matrix $\boldsymbol{B}(\mathcal{G})$.

As shown in Figure 4.6, the duplicated edge pair e_4 and e_5 is detected because they have non-zero weights on the two same rows: row v_3 and v_4. In Figure 4.7, e_5 is removed by deleting the column e_5 and keeping the lesser weight for the remaining edge, i.e. the updated weight of e_4 is assigned as $\min(w_4, w_5) = 3$.

After the short edge collapses and duplicate edge removals, the node types in the updated graph model may change. According to the definition stated in Section 4.1.2, the untagged node v_5 in Figure 4.8 becomes a regular node since there are only two edges connected to it. Changes of node type only appear on the nodes connected to the newly-merged node. These new regular nodes in the graph can be removed using the same way, e.g. v_5 is merged to v_3 and e_4 are deleted. The graph $\mathcal{G}$ is processed as shown in Figure 4.9 into the graph in Figure 4.10.

Note that the topology of a graph may change due to edge collapse. This may leave duplicate edges oriented from the remaining node after the edge collapse. Deleting one edge of the duplicate pair by crossing its corresponding column, and the weight of the remaining one is updated using the lesser value between the weights of the two edges in

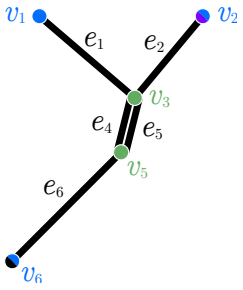


(a) $\mathcal{G}$

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	4	0	0	0	0	0
v_2	0	4	0	0	0	0
v_3	4	4	4	3	4	0
~~v_4~~	~~0~~	~~4~~	~~2~~	~~0~~	~~4~~	~~0~~
v_5	0	0	0	3	4	5
v_6	0	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.5 Removing short edge e_3

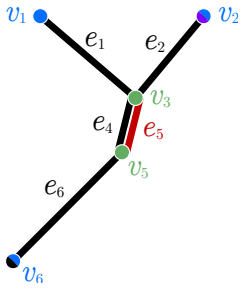


(a) $\mathcal{G}$

	e_1	e_2	e_4	e_5	e_6
v_1	4	0	0	0	0
v_2	0	4	0	0	0
v_3	4	4	3	4	0
v_5	0	0	3	4	5
v_6	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.6 Detecting the duplicate edges e_4 and e_5

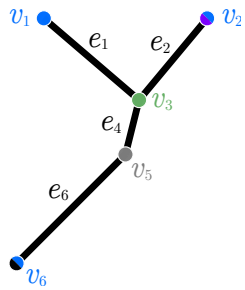


(a) $\mathcal{G}$

	e_1	e_2	e_4	e_5	e_6
v_1	4	0	0	0	0
v_2	0	4	0	0	0
v_3	4	4	3	4	0
v_5	0	0	3	4	5
v_6	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.7 Deleting edge e_5

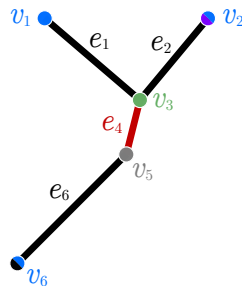


(a) $\mathcal{G}$

	e_1	e_2	e_4	e_6
v_1	4	0	0	0
v_2	0	4	0	0
v_3	4	4	3	0
v_5	0	0	3	5
v_6	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.8 Node v_5 is a regular node

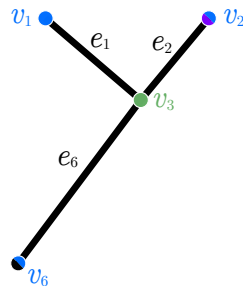


(a) $\mathcal{G}$

	e_1	e_2	e_4	e_6
v_1	4	0	0	0
v_2	0	4	0	0
v_3	4	4	6	5
~~v_5~~	~~0~~	~~0~~	~~3~~	~~5~~
v_6	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.9 Merging node v_5 to node v_3



(a) $\mathcal{G}$

	e_1	e_2	e_6
v_1	4	0	0
v_2	0	4	0
v_3	4	4	5
v_6	0	0	5

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.10 Regular node v_5 is removed

the pair. Furthermore, the deletion of untagged regular voxels may also cause changes in connectivity and duplicate edges may appear again. Thus sequential checks for duplicated edges and untagged regular nodes are adopted to process the graph model until the graph model consists only of unique edges and is free of untagged regular nodes. These checks are systematised as one function, `RecheckGraph`, in Algorithm 4.3. Function `RecheckGraph` loops over edges to search for duplicates and delete them. If there is deletion of edge, thereafter the function immediately goes through nodes to check whether there is any regular node existing. Algorithm 4.3 merges the untagged regular nodes to other nearby nodes if necessary.

To end this section, Algorithm 4.4 is given to describe the approach of merging all short edges in a graph model. In this thesis, the edge collapses are for eliminating edges constructed by close joint voxels to generate a more structurally sound frame model input for frame optimisation. The method of removing edges by manipulating the rows and columns in incidence matrices can be easily generalised for all kinds of short edge.

Algorithm 4.3: Removal of duplicate edges and regular nodes

Input : $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, graph containing duplicate edges or regular nodes
$\boldsymbol{B}(\mathcal{G})$, incidence matrix of $\mathcal{G}$
update : $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, graph model free of duplicate edges or regular nodes
$\boldsymbol{B}(\mathcal{G})$, incidence matrix of the updated $\mathcal{G}$

Function `RecheckGraph`($\mathcal{G}$, $\boldsymbol{B}(\mathcal{G})$)**:**
  /* Check if the graph has duplicate edges */
  while $\exists\ e_m, e_n \in \mathcal{G}$, $(e_m = e_n = \{v_i, v_j\})$ **do**
    w_m: weight of edge e_m
    w_n: weight of edge e_n
    $\boldsymbol{B}(\mathcal{G})_{v_i e_m} \leftarrow \min(w_m, w_n)$
    $\boldsymbol{B}(\mathcal{G})_{v_j e_m} \leftarrow \min(w_m, w_n)$
    delete column e_n from $\boldsymbol{B}(\mathcal{G})$ and e_n from $\mathcal{G}$
    /* Check if the graph has regular nodes after removing edge e_n */
    while $\exists v_r \in \mathcal{G}$, ($v_r$ *is untagged regular node*) **do**
      $e_t = \{v_r, v_s\}$: one of the only two edges connected to v_r
      v_s: the other node of edge e_t than v_r
      row $v_s \leftarrow$ row v_s + row v_r and update connectivities in $\mathcal{G}$
      delete column e_t from $\boldsymbol{B}(\mathcal{G})$ and edge e_t from $\mathcal{E}$
      delete row v_r from $\boldsymbol{B}(\mathcal{G})$ and node v_r from $\mathcal{V}$

Algorithm 4.4: Edge collapse using incidence matrix

input : $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, graph containing short edges
$\boldsymbol{B}(\mathcal{G})$, the associated incidence matrix
update : $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, graph free of short edges
$\boldsymbol{B}(\mathcal{G})$, incidence matrix of the updated $\mathcal{G}$

```
forall edge e_i = {v_j, v_k} ∈ G do
    if v_j is joint node and v_k is joint node then
        w_i: weight of edge e_i        ▷ can be assigned as either B(G)_{v_j e_i} or B(G)_{v_k e_i}
        if w_i = 2 then
            if v_j without tags or v_k without tags then
                row v_j ← row v_j + row v_k and update corresponding
                  connectivities in G
                delete column e_i from B(G) and edge e_i from E
                delete row v_k from B(G) and node v_k from V
                if v_j without tags and v_k without tags then
                    coordinate of v_j ← (coordinate of v_j+coordinate of v_k)/2
                if v_j without tags and v_k with tag_m then
                    coordinate of v_j ← coordinate of v_k
                    tag of v_j ← tag_m
                /* Ensure the graph contains only unique edges and no regular node  */
                RecheckGraph(G,B(G))        ▷ call function in Algorithm 4.3
```

4.1.5 Pruning

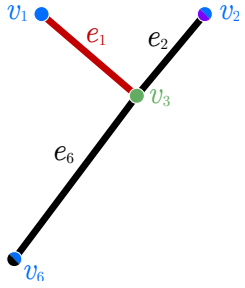
In general, the simplified graph model coming from the skeleton consists of redundant branches due to the asymmetry and small-scale noise on the boundary surface of the thresholded model, as discussed in [44, 119, 139–141]. One can preprocess the voxel chain to clean the noise branches. Extensive research provides efficient skeleton pruning methods, e.g. skeleton filtering [141, 142], geodesic function approaches [97, 143], continuity based approaches [144] and local features focused methods [103, 119, 122, 145, 146]. The literature provides abundant choices in skeleton pre-processing for various applications. Since this thesis focuses on the case where the graph model is used to establish a structural frame, only the useful frame members in load transmission are considered to be kept. Instead of editing the voxel chain, a novel and straightforward way is proposed to prune to-be-frame graph models using incidence matrix.

First the 'useless' branches are located. The branches which lead to structural frame members with zero stress can be safely pruned. Zero-stress frame members can be detected as these are not directly connected to a homogeneous Neumann boundary.

Thus, the edges with at least one end voxel that is not tagged as being loaded nor on such boundary conditions make no contribution to load path are zero-stress members. Their pruning is again implemented with the incidence matrix. As illustrated in Figures 4.11 and 4.12, untagged end node v_1 and its attached edge e_1 can be deleted. Deletion is performed simply by removing the corresponding rows and columns. There is no need to do the row addition to pass the connectivity, since the edge connected to the node-to-delete is also to be deleted.

The pruning may cause changes in graph topology, see Figure 4.12, v_3 turns out to be a untagged regular node when loosing edge e_1. As shown in Figure 4.13 shows that v_3 is processed in the same way as stated in Section 4.1.4, i.e. removing the row v_3 and column e_2 from the incidence matrix in Figure 4.13. Sequential checks for duplicate edges and untagged regular nodes are also required. Figure 4.14 demonstrates the final graph model, which preserves the positions of tagged nodes, v_2 and v_6, and consists of no zero stress members.

The pruning process is described as in Algorithm 4.5. It searches all untagged end nodes. The algorithm then deletes the found ones along with the edges connected to them. Pruning is an optional tool for the user who desires the frame free of zero-stress members. The zero-stress members can also be detected and removed through the frame layout optimisation, but pruning before the optimisation will give a compact frame model with fewer geometric degrees of freedom to be optimised.

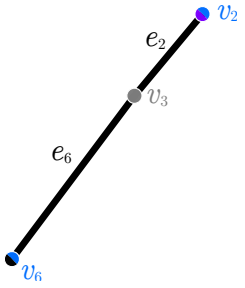


(a) $\mathcal{G}$

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_2 & e_6 \end{array} \\ \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 4 & 4 & 5 \\ 0 & 0 & 5 \end{pmatrix} \end{array}$$

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.11 Deleting zero-stress member e_1

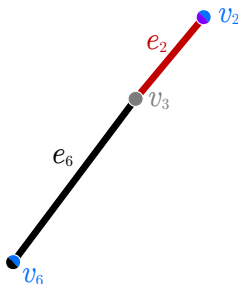


(a) $\mathcal{G}$

$$\begin{array}{c} \\ v_2 \\ v_3 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cc} e_2 & e_6 \end{array} \\ \begin{pmatrix} 4 & 0 \\ 4 & 5 \\ 0 & 5 \end{pmatrix} \end{array}$$

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.12 Node v_3 is a regular node

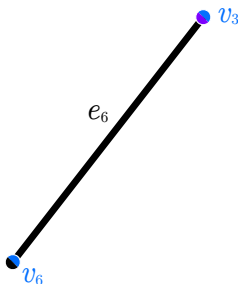


(a) $\mathcal{G}$

$$\boldsymbol{B}(\mathcal{G}) = \begin{array}{c} \\ v_2 \\ v_3 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cc} e_2 & e_6 \end{array} \\ \begin{pmatrix} 8 & 5 \\ 4 & 5 \\ 0 & 5 \end{pmatrix} \end{array}$$

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.13 Removing regular node v_3



(a) $\mathcal{G}$

$$\boldsymbol{B}(\mathcal{G}) = \begin{array}{c} \\ v_3 \\ v_6 \end{array} \begin{array}{c} \begin{array}{c} e_6 \end{array} \\ \begin{pmatrix} 5 \\ 5 \end{pmatrix} \end{array}$$

(b) $\boldsymbol{B}(\mathcal{G})$

Fig. 4.14 Final graph

Algorithm 4.5: Pruning using incidence matrix

input : $\mathcal{G}$, graph model with possible zero-stress members
$\boldsymbol{B}(\mathcal{G})$, incidence matrix of $\mathcal{G}$
update : $\mathcal{G}$, graph model without zero-stress members
$\boldsymbol{B}(\mathcal{G})$, incidence matrix of the updated $\mathcal{G}$

1 **forall** *node* $v_j \in \mathcal{G}$ **do**
2 **if** v_j *is untagged end node* **then**
3 $e_i = \{v_j, v_k\}$: the only edge connected to v_j
4 v_k: the other node of edge e_i than v_j
5 delete column e_i from $\boldsymbol{B}(\mathcal{G})$ and edge e_i from $\mathcal{G}$
6 delete row v_j from $\boldsymbol{B}(\mathcal{G})$ and node v_j from $\mathcal{G}$
 /* Ensures the graph contains only unique edges and no regular node */
7 `RecheckGraph`($\mathcal{G}$,$B(\mathcal{G})$) ▷ call function in Algorithm 4.3

Appendix E covers more frame extraction examples. In these examples, the graph models are with more tags.

4.2 CAD models generation

The graph model supplemented by the node coordinates and member cross-sections provides sufficient information to generate a compact structural frame model. It is more straightforward to create a parametric solid CAD model from a structural frame

model than from a solid voxel model. Once an optimal frame model is obtained, there exist various ways to convert the frame model into a compact CAD model. For instance, one can use a *subdivision surface* model [147] or *Boundary representation* (B-rep) model. The shapes of these two geometric representations can be conveniently edited by manipulating control points. Both are popular geometric representations in CAD systems [148]. When using either of these as desired CAD output, one should be aware of three cases, listed below, which may lead to problematic models.

Case 1: Joints are close to each other while the cross-sectional areas of the members between these joints are relatively large. This will cause the intersections between the spline patches or meshes generated around these two joints. Removing short members in advance using edge collapse techniques, see Sections 4.1.4 and 2.2.3, can avoid this intersection issues.

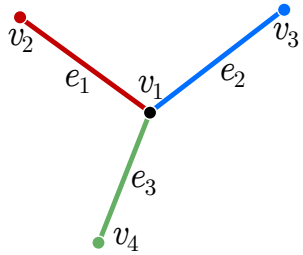
Cases 2: The edges collide with each other, or even the edges are not in contact, the volumes of the members collide. Considering the CAD output of the proposed approach in this thesis will be handed back to designers and engineers, such collision-related problems are left to them. With the CAD model, one can edit shapes in straightforward ways. Alternatively, one can also introduce penalisation contact forces into frame optimisation process to avoid beam collisions [149, 150]. Note that this method can not detect the collision between two closely parallel beams.

Case 3: A large number of edges are connected at one joint. This issue, particularly when subdivision surfaces are the output, can cause unnecessarily fine spline patches or meshes to approximate the high-order shape generated around the joint. One possible solution is to optimise the number of members linking at the same joint, e.g. by splitting the joint into two or more and spreading the connected members to these split joints. Another solution is to use *QUADOR* proposed by Gupta et al. [151, 152], which gives exact analytical lower-order (quadratic) solutions of intersection curves to generate coarse spline patches/meshes. For the one pursuing fillet around the joint (for better manufacturability and ease of stress concentration), please refer to the improved QUADOR developed by Cirak and Sabin [153].

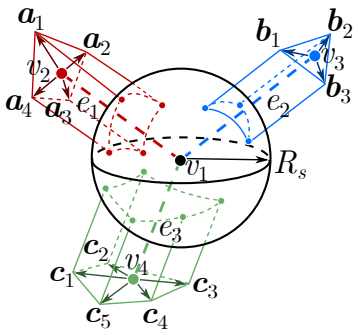
4.2.1 Reconstruction as structured surface meshes

The surface mesh is desired to be as coarse as possible when being imported into CAD systems or used as the mesh for isogeometric shell analysis and shape optimisation [154, 13].

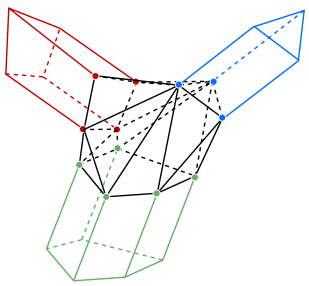
Here a method is proposed to generate the coarse triangular surface mesh based on the topology and member sizes of the frame. As illustrated in Figure 4.15, It is assumed that a frame has one joint node v_1 which reaches nodes v_2, v_3 and v_4 through edges e_1, e_2 and e_3 respectively. The frame members are described with arbitrary cross-section shapes and such cross-sections are acquired by offsetting the edges through vectors from the beam node to vertices of the cross-section. These vectors are user-defined, and they reflect the user's expectation of the alignments of cross-sections and their shapes.



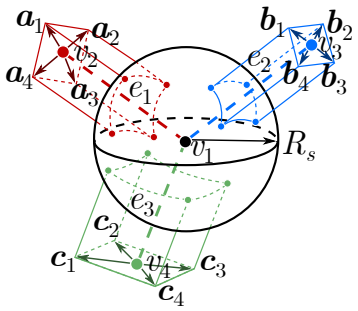
(a) Graph model



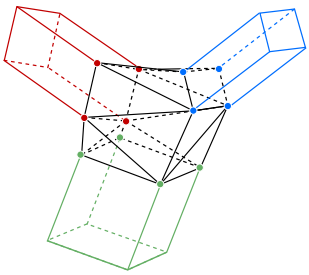
(b) Intersecting offset lines with sphere



(c) Triangular mesh



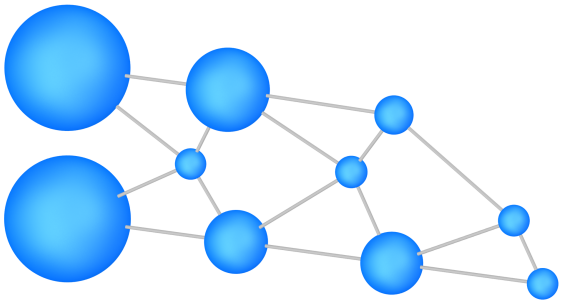
(d) Intersecting offset lines with sphere



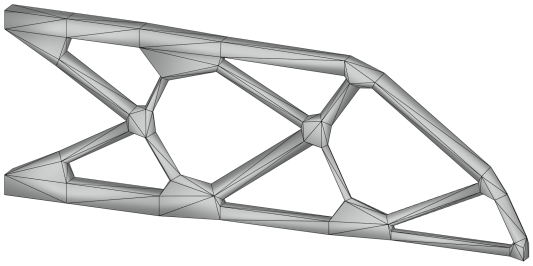
(e) Triangular mesh

Fig. 4.15 Generating triangular mesh using convex hull. For beams using arbitrary cross-section shapes, the intersection points between the sphere and offset lines are shown on the sphere in (b). Then the triangular mesh is generated for the convex hull of these intersections at the node v_1. Such mesh is stitched to the meshes of the frame members in (c). (d) and (e) show a simple case where cross-sections are rectangles.

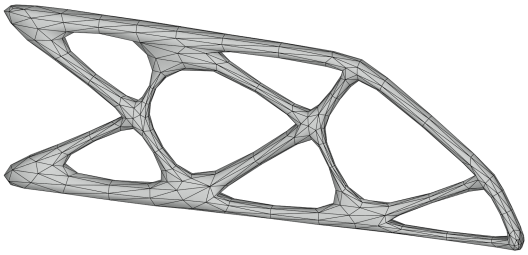
For example, in Figure 4.15d, four red lines are generated by offsetting red edge e_1 along vectors $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$ and $\boldsymbol{a}_4$, since e_1 has a quadrilateral cross-section. Similarly the other two edges e_2 in blue and e_3 in green are offset. Doing this gives 12 intersection points (shown as points on the sphere in Figure 4.15d) of the offset lines and the sphere with a given radius of R_s. R_s is also a user-defined parameter which interprets the scale of the joint design. The proper radius R_s can avoid the generated meshes to be self-intersected. The sphere is convex, which indicates there is always one unique convex hull containing all these intersection points on the spherical surface [155]. The convex hull and its triangular mesh can be seen in Figures 4.15c and 4.15e. Such mesh around the joint can be easily connected to the meshes of frame members. An example of using the proposed surface generation method is shown below in Figure 4.16. The resulting mesh is a compact triangular surface mesh shown in Figure 4.16b.



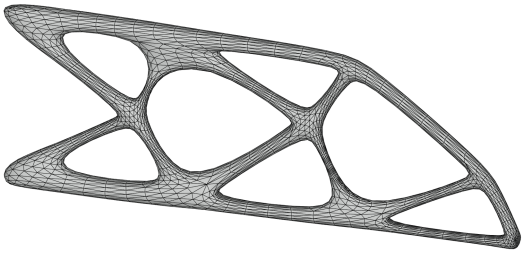
(a) Graph model with spheres of given radii



(b) Triangular mesh generated from (a)



(c) Refined (a) with one iteration



(d) Refined (b) with two iterations

Fig. 4.16 An example to illustrate the triangular mesh generation using convex hull. All members are assumed to have rectangular cross-sections. The blue spheres in (a) are centred at the frame joints with the given radii. Stitching the frame meshes to triangular meshes of convex hulls at joints gives the mesh in (b). Using Loop subdivision scheme to refine (b) leads to (c) with one iteration and (d) with two iterations.

There are similar methods of generating quadrilateral meshes. For instance, *B-mesh*, implemented by Li et al. [156], is a modelling approach which begins with the input of curve skeleton and spheres centred at the joints with given radii. The B-mesh method then interpolates spheres along the edges between two corresponding spheres at the

nodes. The intersection points between these spheres are consequently used to generate the quadrilateral meshes also using convex hull. Current work on B-mesh can only generate manifold mesh from a tree graph. Simple structural frames without self-loops can be constructed as a compact structured mesh using this approach. Similarly, the concept of *skeleton scaffolding* proposed by Panotopoulou et al. [157] is to generate a as coarse structured quadrilateral mesh as possible. It is also based on the curve skeleton and given spheres at the curve skeleton joints. They partition the joint sphere into quadrangles and construct sleeves around skeleton edges. The resulting coarse mesh is then constructed with the minimum twisting. This approach can work on a graph model with self-loops. These two approaches, along with the proposed triangular mesh generation method, are recommended for the user who expects a coarse structured mesh as the output.

4.2.2 Reconstruction as solid using CSG tree

Binary CSG tree, or briefly CSG tree, is a technique used to generate solid models. It models solids even of complicated shape based on iterative boolean operations of simple geometric primitives [158]. The generation of geometric primitives is in B-rep. Figure 4.17 shows a CSG tree, which yields a tri-tubular shape solid model. The construction begins with the generation of one sphere and one cylinder, followed by a union operation of the cylinder and the sphere, then the unionised shape is further added to another cylinder. The process continues until the target shape is achieved.

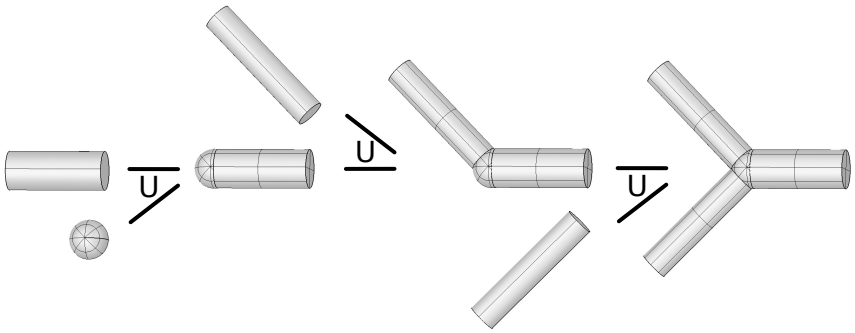


Fig. 4.17 Illustration of the CSG tree to generate a tri-tubular CAD model. The CAD model is generated by successively combining cylinders with spheres using boolean union operations.

In this thesis, it is assumed that there is a one-to-one correspondence between nodes and edges of the graph model and joints and members of the frame model. Besides, members are assumed to be only subjected to end forces and moments, i.e.

no distributed loads, so that they can be straight and have uniform cross-sections along their lengths. For simplicity, all cross-sections are assumed to be circular. Note that these assumptions are fully justified if there is no distributed loading in topology optimisation. These restrictions can mostly be relaxed if necessary. Thus, the beam members can be modelled using cylinder primitives with beam central axes as cylinder central axes and beam cross-sectional radii as cylinder cross-section radii. Spheres are placed at the frame joints to achieve watertight connections between cylinders in the CSG tree. The radius of the sphere is chosen as the maximum radius of all cylinders connected to the joint. In order to avoid trimming errors, a factor of 1.05 is used to multiply to the maximum radius.

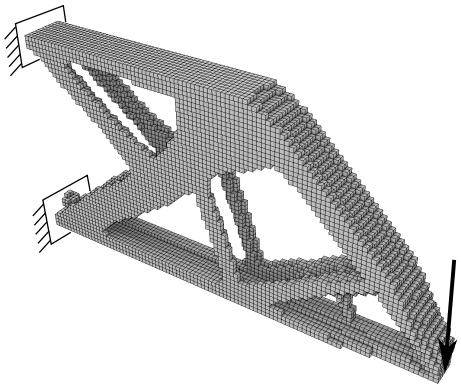
Alternatively, a somewhat similar approach was used by Smith et al. [159, 160] to generate CAD models for optimised truss structures. In their method, the idea is to connect two of the most stressed members continuously across a joint or to add fillets to reduce stress concentrations.

4.3 CAD generation example: MBB beam

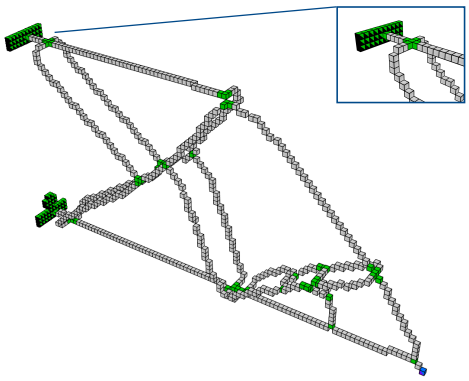
In this section, an MBB beam example is used to illustrate the process of frame extraction and CAD model generation. It is assumed that the MBB beam has been topology-optimised and thresholded to the voxel model shown in Figure 4.18a. Its skeleton is shown in Figure 4.18b.

The voxel types are identified and coloured in the same fashion as in previous sections, i.e. joint voxels/nodes are marked with green, end voxels/nodes with blue and regular nodes/voxels with grey, and those with tags are half-coloured in black for voxels/nodes with loadings or purple for voxels/nodes with boundary conditions. As can be seen in the blue window in Figure 4.18b, the clustered tagged voxels are all identified as tagged joint voxels. Using Algorithm 4.2 can simplify the cluster of tagged voxels to one representative tagged voxel as shown in the blue window in Figure 4.18c.

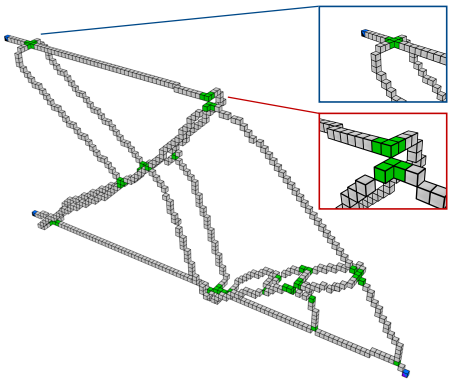
The voxel chain model, without the confusion caused by the clusters of tagged voxels, is converted into the graph model in Figure 4.18d where nodes $(v_1, v_2, v_3, v_4, v_5)$ and edges (e_1, e_2, e_3, e_4) are used to illustrate the CSG tree; R_1, R_2, R_3, R_4 are the cross-section radii of the corresponding edges. As can be seen in the red window in Figure 4.18c, six joint voxels are in close proximity. Thus short edges are constructed from these voxels. Algorithm 4.4 is used to remove all these two-voxel-long edges. Edge collapses for these short edges produce the graph model shown in Figure 4.18d with critical load path presented. In the red window in Figure 4.18d, only one joint voxel



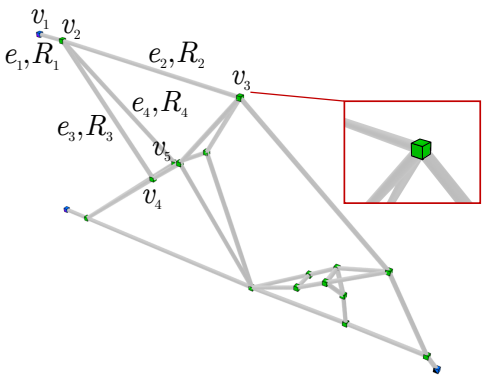
(a) Voxel model



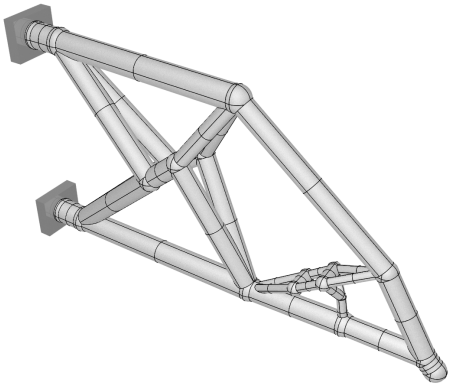
(b) Skeleton model



(c) Removed redundant tagged voxels



(d) Graph model extracted from (c)



(e) CAD construction using CSG tree

Fig. 4.18 CAD model generation for a topology-optimised MBB beam. The topology-optimised model (a) is skeletonised into the voxel model (b). Removing redundant tagged voxels in (b) gives (c), then a frame model from (c) is constructed in (d). Final CAD model constructed using the CSG tree with the frame model and the corresponding cross-sectional information.

remains. There are three end voxels in the graph, and they are all tagged as either loaded or with boundary conditions. Therefore, no zero-stress member is detected and no pruning is needed.

The first 11 steps of the CSG tree are listed in Table 4.1. With these 11 steps, all edges connected to v_0 and v_1 are constructed as cylinders and unionised with joint spheres as a CAD model. The steps in the rest of the CSG tree follow the same styles, i.e. generating spheres, cylinders and the unions of them. The CAD model constructed by the CSG tree in Table 4.1 can be seen in Figure 4.18e. To this end, the CAD system Rhino3D [161] or FreeCAD [162] is used in this thesis. However, the proposed approach can be realised in any CAD system which provides a scripting interface or API (Application Programming Interface).

Table 4.1 Procedure of CSG tree for CAD model generation of the MBB beam

Step	Operation
Step 1	Generate **object 1** as a sphere with the centre = v_1 and radius = $\max\{R_1\}$
Step 2	Generate **object 2** as a cylinder with the centre axis = $e_1 = \{v_1, v_2\}$ and radius = R_1
Step 3	Generate **object 3** as the union of **object 1** and **object 2**
Step 4	Generate **object 4** as a sphere with the centre = v_2 and radius = $\max\{R_1, R_2, R_3, R_4\}$
Step 5	Generate **object 5** as the union of **object 3** and **object 4**
Step 6	Generate **object 6** as a cylinder with the centre axis = $e_2 = \{v_2, v_3\}$ and radius = R_2
Step 7	Generate **object 7** as the union of **object 5** and **object 6**
Step 8	Generate **object 8** as a cylinder with the centre axis = $e_3 = \{v_2, v_4\}$ and radius = R_3
Step 9	Generate **object 9** as the union of **object 7** and **object 8**
Step 10	Generate **object 10** as a cylinder with the centre axis = $e_4 = \{v_2, v_5\}$ and radius = R_4
Step 11	Generate **object 11** as the union of **object 9** and **object 10**
⋮	⋮

CHAPTER 5

Applications

This chapter provides the numerical studies of the proposed approach. To begin with, Section 5.1 reviews the sequence of steps from the definition of a topology optimisation problem to obtaining a structurally sound parametric CAD geometry. It is followed by additional detail for each step and the interplay between steps. Accordingly, in order to test the proposed workflow, Section 5.2 gives four examples including one 2D topology optimisation benchmark in Section 5.2.1, and three 3D cases to illustrate the automated process of the proposed approach in Sections 5.2.2, 5.2.3, and 5.2.4. These four examples illustrate the entire work flow from a topology-optimised geometry, to a skeleton model, then to a size and layout optimised frame model and finally to a compact CAD model.

5.1 Overall workflow

From the input of design requirements to the output of a compact parametric CAD model, the five steps of the proposed approach are listed as follows:

Step 1: Topology optimisation

It is assumed that the finite element mesh for the optimisation problem (2.1) is a structured hexahedral grid. In most optimisation problems, the design domain is a parallelepiped which can be discretised with a structured grid. Other more complex design domains can be considered by computing their implicit, or level set, representation and embedding them in a structured hexahedral grid, see the horse mesh and quadcopter CAD model examples in Figures 3.20 and 3.26a. From the outset, the

voxels outside the design domain are chosen as void by setting their Young's modulus with $E_{\min}$.

Step 2: Skeletonisation

The input to the homotopic skeletonisation algorithm is a binary image defined on a hexahedral structured grid. The binary image is obtained by thresholding the topology-optimised geometry. The threshold value η is chosen so that the prescribed material volume constraint (2.1c) in topology optimisation is satisfied. The output of the skeletonisation described in Algorithm 3.1 is a curve skeleton in the form of a voxel chain with the same topology as the topology-optimised geometry.

Step 3: Structural frame model generation

The topology and joint coordinates of the frame model are extracted from the voxel chain model. It is assumed that all the members are straight, have a circular cross-section with the same diameter and their total volume is equal to the prescribed material volume $V_f\overline{V}$ in topology optimisation. It is possible to obtain frame models with more complex geometries and non-uniform cross-sections. This appears to be, however, usually to be undesirable from an ease and cost of manufacturability viewpoint.

Step 4: Frame size and layout optimisation

The structural frame model extracted from the skeleton is usually suboptimal. That is, the compliance of the structural frame is larger than that of the topology-optimised geometry. This is unavoidable given that skeletonisation does not involve any structural design considerations. The boundary conditions and loadings applied on the frame are identical to those in topology optimisation. Subsequently several steps of sequential size (2.22) and layout optimisation (2.23) are applied to recover the optimality of the frame structure. In size optimisation the cross-section area A_i of the circular member cross-sections and in layout optimisation the coordinate components s_m of the joints are updated. Both optimisation problems are solved with the SQP (sequential quadratic programming) method [163]. Throughout optimisation, the volume of the frame is constrained to be equal to the prescribed material volume in topology optimisation $V_f\overline{V}$. It is straightforward to consider additional constraints pertaining to the member cross-sections, the positions of the joints, or positions and orientations of the members. The first step is always size optimisation on the frame with the uniform cross-sectional area, attempting to achieve a shape similar to the thresholded topology-optimised

solid model, which is followed by as many as necessary alternating layout and size optimisation steps until the compliance cost function is converged. If the length of any member reduces to unnecessarily small during shape optimisation, the member is removed, its end nodes are merged, and the iteration continues.

Step 5: CAD model generation

A compact CAD model of the structural frame can essentially be generated in any parametric CAD system using a fully automated process. The members are represented by cylinders and the joints by spheres which are combined by boolean operations. The underlying binary CSG tree representation makes it easy to edit further and to refine the optimised design.

5.2 Examples

All four examples in this section use the same material with properties listed in Table 5.1.

Table 5.1 Material properties for frame optimisations

Solid Young's modulus $\overline{E}$	Void Young's modulus $E_{\min}$	Poisson ratio ν
2.1×10^5	10^{-9}	0.3

5.2.1 Cantilever

To start with, a relatively simple example is considered here, which is a cantilever plate. The cantilever plate shown in Figure 5.1 is one of the most widely studied benchmark examples in topology optimisation, see e.g. [56].

Problem description

The size of the design domain is $150 \times 50 \times 4$. The left face of the domain is clamped while all other faces are free. A point force with a value of $F = 100$ is applied at the centre of the mid-right face.

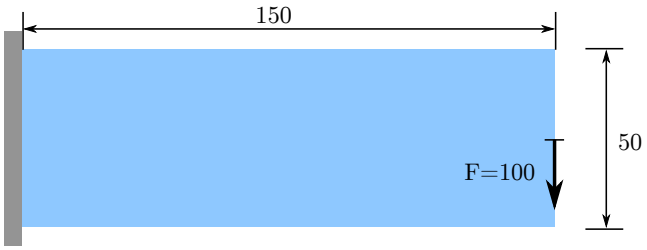


Fig. 5.1 Cantilever: geometry, boundary conditions and loadings. The design domain is in blue and is fixed to the edge in grey.

Step 1: Topology optimisation

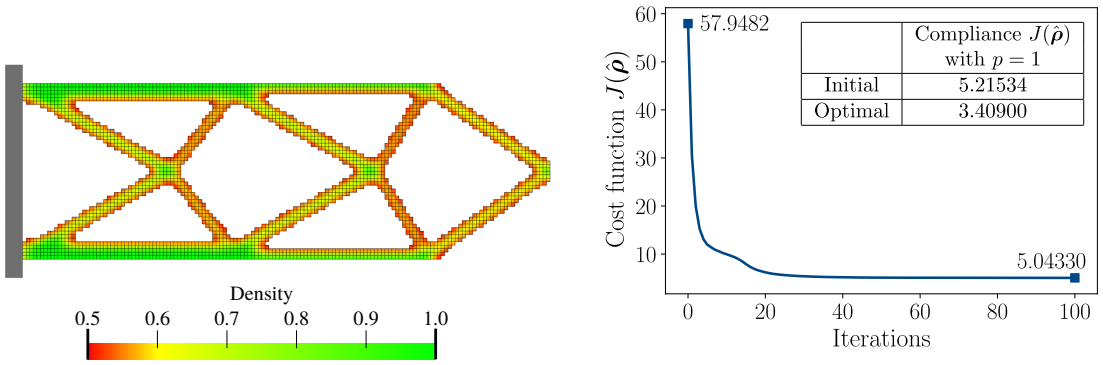
The finite element discretisation consists of $150 \times 40 \times 4$ linear hexahedral elements. The cost function to be minimised is the compliance $J(\hat{\boldsymbol{\rho}})$. The parameters for the topology optimisation are listed in Table 5.2. The topology optimisation starts with the initial shape of uniform density of 0.3.

Table 5.2 Cantilever: topology optimisation parameters

Penalisation power p	Filter radius R	Volume fraction V_f	Maximum iterations
3	3	0.3	100

The cost function $J(\hat{\boldsymbol{\rho}})$ drops from 57.9482 to 5.04330 during the topology optimisation. After the optimisation, penalisation power is set to $p = 1$ in (2.2) to make cost function turn into structural compliance to be compared with the compliance of the frame structure. Hereinafter, for the sake of clarity, the value of $J(\hat{\boldsymbol{\rho}})$ with the user-defined penalisation power is referred as *cost function value*, whereas the value of $J(\hat{\boldsymbol{\rho}})$ with $p = 1$ is referred as *compliance*. The final shape gives the compliance $J(\hat{\boldsymbol{\rho}}) = 3.40900$. Figure 5.2a depicts the optimised cantilever structure with only the voxels above a relative density $\eta = 0.5$ shown. Since the model after thresholding will be regarded as a binary image, all voxels on such model are regraded as solid voxels. The value $\eta = 0.5$ suggested here gives a thresholded mesh with 9348 voxels, which is approximately the volume constraint $V = V_f\overline{V} = 9000$. The shape in Figure 5.2a gives the user an intuitive expectation of the sizes of the final frame model. One can iteratively determine η to threshold the voxel model to a geometry with volume even closer to $V_f\overline{V}$, but doing such is not recommended due to the expensive computing.

Since the frame optimisation will recover the loss in compliance and the aim of this thesis is to introduce the proposed CAD model generation work flow, less interest is



(a) Thresholded geometry with $\eta = 0.5$ (b) Convergence

Fig. 5.2 Cantilever: topology optimisation. Thresholded model (a) has 9348 voxels. The table in (b) gives the initial and final structural compliances ($J(\hat{\boldsymbol{\rho}}), p = 1$).

laid in pursuing highly accurate topology-optimised shape. In this case, there is no need to exhaust the topology optimisation to get the shape with global minimum cost function value. Thus, the topology optimisation is terminated at the 100th iteration. Further proof of the propriety of doing this can be viewed in Figure 5.2b. The cost function value merely changes after the 50th iteration. The reason for the difficulty in cost function converging is discussed in Section 2.1.6.

Step 2: Skeletonisation

In this chapter, for convenience, skeletonisation and skeleton are referred to as curve skeletonisation and curve skeleton respectively.

As the first step in obtaining the structural frame model from the voxel model in Figure 5.2a, the homotopic skeletonisation algorithm is employed. The skeletonisation algorithm yields after 5 voxel removal steps the voxel chain model shown in Figure 5.3. Note that one removal step consists of six sub-steps of six deletion directions, see Section 3.2.2. The number of voxels on voxel chain is 523, which is 5.6% of the voxel number of the topology-optimised model. As is visually evident, the voxel model in

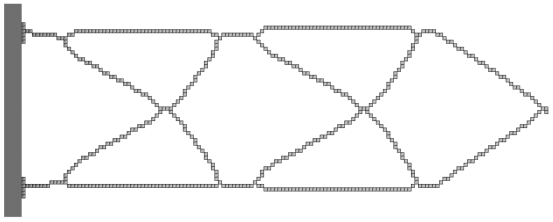


Fig. 5.3 Cantilever: structural skeleton. The skeleton has 523 voxels.

Figure 5.2a and the voxel chain model in Figure 5.3 have the same topology, i.e. both of them have one connected object and six holes. Besides, with the voxels on the left edge and mid-right edge tagged, the skeletonisation keeps the locations of boundary conditions and loadings.

Step 3: Frame model generation

The voxel chain model is converted to the structure depicted in Figure 5.4a by first identifying the 16 joints which are featured voxels[1] using the definition stated in Section 4.1.2 and then connecting them with 21 members.

Step 4: Frame size and layout optimisaiton

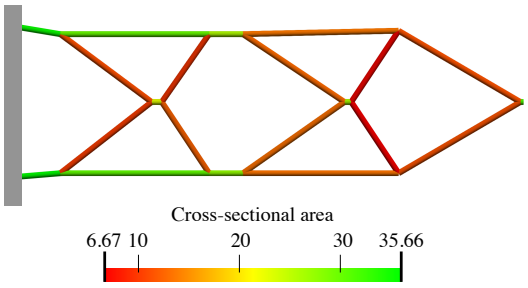
The parameters for the frame optimisation are given in Table 5.3. There is no lower/upper bound for cross-section areas in this example. To recap, the termination criterion of sequential size and layout optimisation is introduced in 2.2.3. During the layout optimisation, a member shorter than merge ratio ζ of the total length of all members sharing the same node is considered to be a short member, and its two end nodes are merged.

Table 5.3 Cantilever: frame optimisation parameters

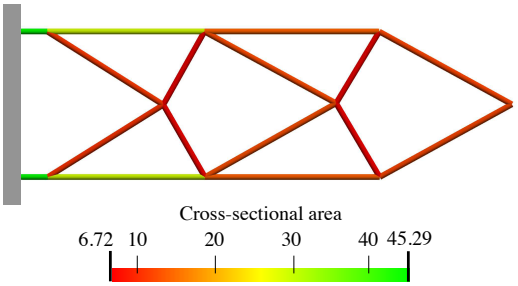
Minimum area $A_{\min}$	Maximum area $A_{\max}$	Merge ratio ζ	Tolerance ϵ_{frame}	Volume constraint
–	–	1/20	10^{-4}	$V \leq 9000$

Initially all members are assumed to have the same cross-section areas of $A = 16.240$ giving the total volume $V = 0.3\overline{V} = 9000$. In determining the total volume, only the cross-section areas of beams and their lengths between the joints are considered as stated in (2.30). As is evident from Figure 5.4a, during the conversion to the frame model the optimality of the voxel model is, as expected, compromised; for instance, the non-straight top and bottom members. As seen in Figure 5.4c, this compromise leads to the frame consisting of beams with uniform cross-section areas having compliance $J(\mathbf{A}, \mathbf{s}) = 4.59841$, which is higher than the compliance of the topology-optimised geometry of $J(\hat{\boldsymbol{\rho}}) = 3.40900$. The stretch strain energy and the bending strain energy of the uniform frame are 2.02202 and 0.494215. The minimum and maximum axial

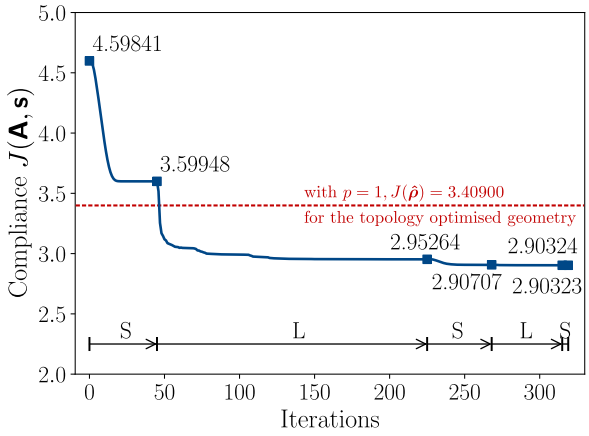
[1] Featured voxels, as defined in Section 4.1.2, are neither regular voxels (with only two 26-neighbours) nor being tagged.



(a) Size optimised frame (initial)



(b) Size and layout optimised frame (final)



(c) Convergence

Fig. 5.4 Cantilever: frame size and layout optimisation. The initially size-optimised frame model (a) has 16 joints and 21 members while the final frame model (b) has 11 joints and 16 members. Red dashed line in (c) shows the structural compliance (3.40900) of the topology-optimised shape.

stress in the cantilever are 0 and 20.9274 respectively. The average axial stress is 8.04859 with a variance of 39.2004.

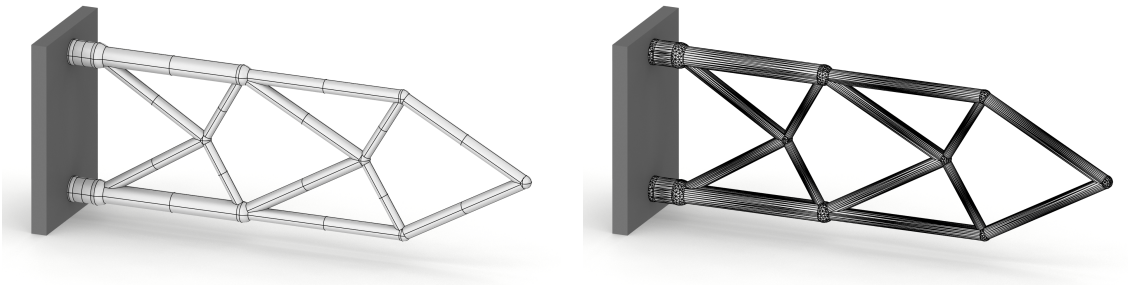
The loss of optimality can however be recovered by optimising the joint positions, i.e. layout optimisation, and the cross-sectional areas, i.e. size optimisation, of the frame. After the first size optimisation step, the compliance is reduced to $J(\mathbf{A}, \mathbf{s}) = 3.59948$ and the subsequent layout optimisation step to $J(\mathbf{A}, \mathbf{s}) = 2.95264$, see Figure 5.4c. Several more steps of size and layout optimisations do not lead to a significant reduction in compliance. Note that the obtained final compliance $J(\mathbf{A}, \mathbf{s}) = 2.90323$ is significantly lower than the compliance $J(\hat{\boldsymbol{\rho}}) = 3.40900$ of the topology-optimised voxel model. The stretch strain energy and the bending strain energy of the final frame structure are 1.42114 and 0.0190822 respectively. The minimum and maximum axial stress in the cantilever are 7.00854 and 8.46234 respectively. The average axial stress is 8.09638

with a variance of 0.199892. As a result, in the final optimised cantilever the total strain energy is reduced, i.e. the structure becomes stiffer, and the frame becomes stretch-dominant. In addition, the stresses are distributed more uniformly in the frame, as happens in the examples in Section 2.2.5.

The cantilever converges at the shape containing 11 joints and 16 beam members. During the layout optimisation, five edges are collapsed. These are the five shortest edges in Figure 5.4a.

Step 5: CAD mode generation

Finally, in Figure 5.5 the solid CAD model of the frame and a faceted triangular STL mesh exported from FreeCAD [162] are shown. To recall, STL file uses piece-wise triangulated surfaces to approximate the trimmed NUBRS surfaces in a CAD model. The lines in the CAD model show the control meshes of the trimmed NURBS patches. To some extent, the shapes of the CAD models in Figure 5.5 generated from the optimal frame are similar to that of the topology-optimised model in Figure 5.2a.



(a) IGES model (b) STL model

Fig. 5.5 Cantilever: parametric CAD model. IGES model (a) is the union of 11 spheres and 16 cylinders. STL mesh (b) has 3430 triangular facets.

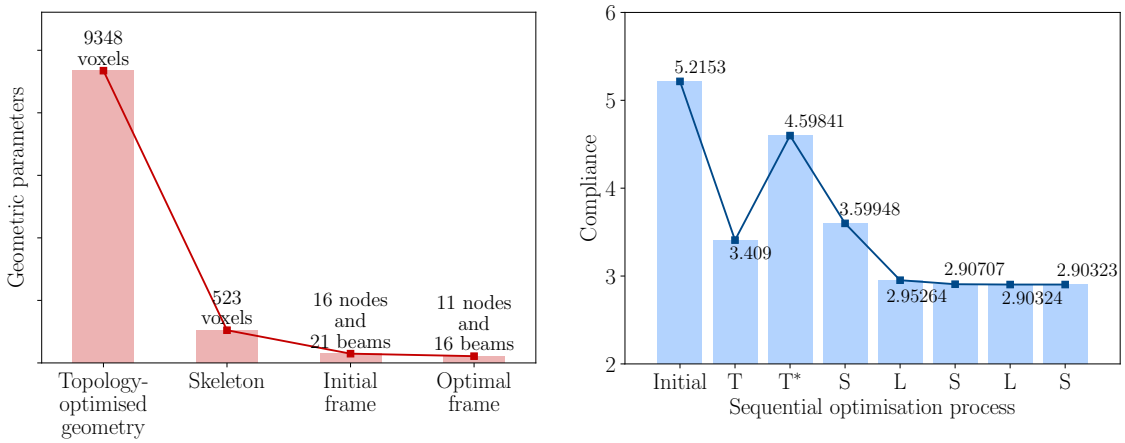
Summary

In order to summarise the results in this example, the changes in the number of necessary geometric parameters and compliances during the workflow are illustrated in Figure 5.6.

As can be seen in Figure 5.6a, the topology-optimised solid model contains 9348 voxels, then the skeletonisation simplifies the solid model into a voxel chain model with 523 voxels. The frame extracted from the voxel chain includes only 16 nodes and 21 beams and the final frame can be defined using even fewer parameters: 11 nodes and

16 beams, which has 0.29% of the number of geometric parameters to represent the topology-optimised model (9348 voxels).

Moreover, in the meantime, the proposed workflow preserves the optimality discovered by the topology-optimisation. Figure 5.6b shows that the final optimal spatial frame with the compliance of 2.90323 which arrives at a similar level of the compliance of the topology-optimised model with the compliance of 3.40900. This cantilever example confirms that the proposed workflow not only significantly simplifies the geometry but also preserves the optimality.



(a) Number of geometric parameters (b) Structural compliance

Fig. 5.6 Cantilever: the change in the number of geometric parameters (a) and compliance (b) during the workflow. The abbreviations In (b) are Initial: the initial model with uniformly distributed density, T: topology optimisation, T*: the frame with uniform-sized beams, S: frame size optimisation and L: frame layout optimisation.

5.2.2 Pipe bracket

As a structure with a truly three-dimensional load path, the pipe bracket shown in Figure 5.7 is considered. The design domain of the cantilever plate in Section 5.2.1 is a cuboid, therefore the upper and lower bound for the frame joint coordinates can be prescribed in a straight forward way, i.e. the joints are constrained to move within a bounding box. For this example, ithe design domain is prescribed with two cylindrical holes (for pipes passing through) using implicit geometric representation, i.e. the level-set function as discussed in Section 2.2.4.

Problem description

The design domain has the size of $120 \times 40 \times 60$ and contains two openings. These openings, each with radius of 18, are for two pipes passing through the domain. Within the openings each pipe is supported at four points, applying at each support a vertical force of $F = 100$ to the domain. The four vertical outer edges of the design domain are chosen to be fixed. The same material properties as in the cantilever plate example are used.

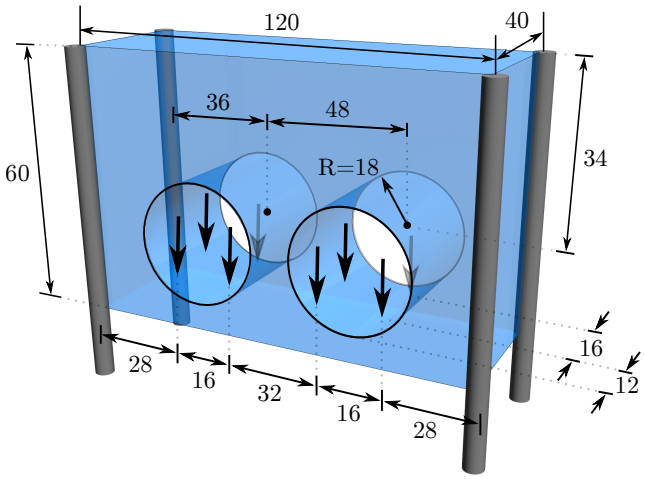


Fig. 5.7 Pipe bracket: geometry, boundary conditions and loadings. The design domain is in blue and fixed to the four vertical grey columns.

Step 1: Topology optimisation

The finite element discretisation consists of $120 \times 40 \times 60$ linear hexahedral elements. Topology optimisation parameters are provided in Table 5.4. To represent the two openings in the finite element model the Young's modulus of the elements within the void cylindrical regions are prescribed with $E_{\min}$.

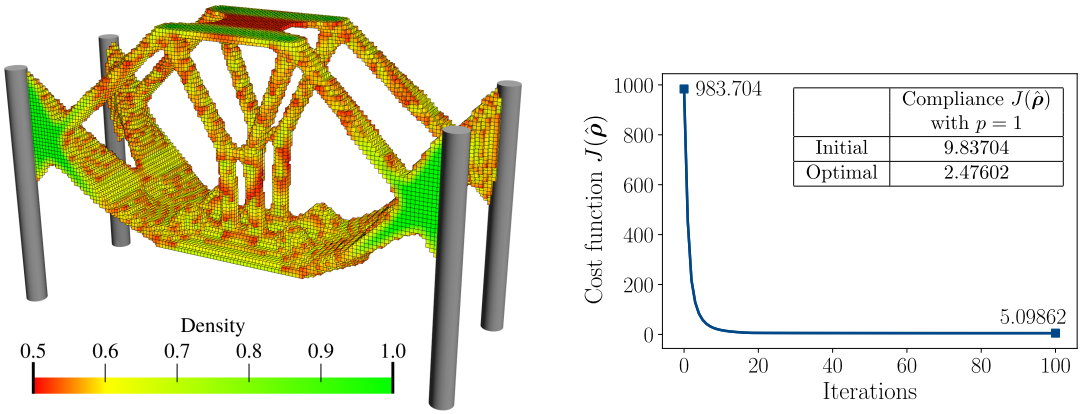
Table 5.4 Pipe bracket: topology optimisation parameters

Penalisation power p	Filter radius R	Volume fraction V_f	Maximum iterations
3	3	0.1	100

As can be seen in Figure 5.8b, the cost function value of the initial shape, with all element densities of 0.1, is 983.704. The high initial cost function value is due

to the small initial density. After the topology optimisation, the cost function value decreases to 5.09862. The final structural compliance of the optimised voxel model is $J(\hat{\boldsymbol{\rho}}) = 2.47602$. Figure 5.8a shows the optimised structure with only the voxels above a relative density $\eta = 0.5$ and the thresholded voxel model has the volume of $26940 \approx 28800 = V_f \overline{V}$.

As easily recognisable from Figure 5.8a, the optimised structure is a combination of an arch-like and a cable-like structure with nearly vertical tension members between the two. In previous example of cantilever, the topology-optimised shape is truss-like, see Figure 5.2a.



(a) Thresholded geometry with $\eta = 0.5$ (b) Convergence

Fig. 5.8 Pipe bracket: topology optimisation. Thresholded model (a) has 26940 voxels. The table in (b) gives the initial and final structural compliances $(J(\hat{\boldsymbol{\rho}}), p = 1)$.

Step 2: Skeletonisation

The skeletonisation algorithm uses 6 removal steps to skeletonise the model of 26940 voxels in Figure 5.8a to a voxel chain model of 937 voxels.

The plate-like parts on the top and bottom of the voxel model are also skeletonised into one-voxel thick chains, which confirms that the skeletonisation algorithm can work not only with the truss-like or frame-like shapes, but also plate-like shapes. Eight tiny branches stretching upwards from the bottom part of the chain connect to the tagged voxels, where concentrated point forces are applied. The voxel chain model has the topology identical to the voxel model and provides a faithful representation of all the load paths that are present in the optimised structure.

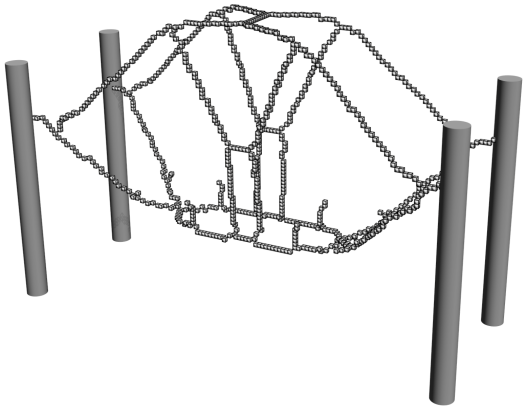


Fig. 5.9 Pipe bracket: structural skeleton. The skeleton has 937 voxels.

Step 3: Frame model generation

As can be seen in Figure 5.9, different from the skeleton in the cantilever example, most skeleton branches, especially the four longest branches, are rather curved than straight. Thus, alternatively, the 50 members can be represented with curves, e.g. polynomials or Bézier curves, which would lead to a structural frame with curved beam members. Although curves resemble closer to the shape of the structural skeleton, this has not been accepted in the proposed approach because such members usually lead to an increase in manufacturing costs and are not preferred by manufacturers. To illustrate this, the Bézier curve fitted frame[1] is shown in Figure 5.10. Manufacturing such curved frame is not practical. As a result, the voxel chain model is converted to the frame structure in Figure 5.11a with 50 straight members and 42 joints.

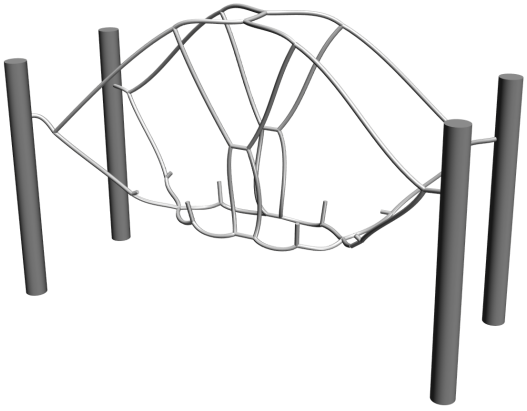


Fig. 5.10 Pipe bracket: Bézier curve fitted structural skeleton.

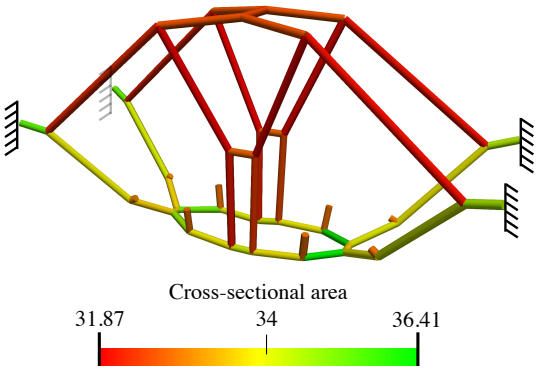
[1]The Bézier curves connected to the joints are generated with the minimum least square root error of distances of the voxels to the Bézier curve [164, 165].

Step 4: Frame size and layout optimisation

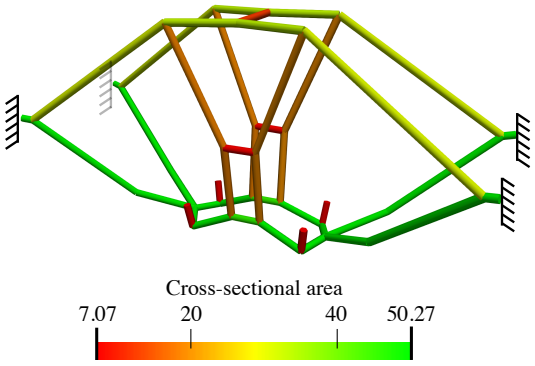
Frame optimisation is undertaken using the parameters in Table 5.5. The geometric constraint is formulated using the level set function, computed using openVDB [89], for the design domain.

Table 5.5 Pipe bracket: frame optimisation parameters

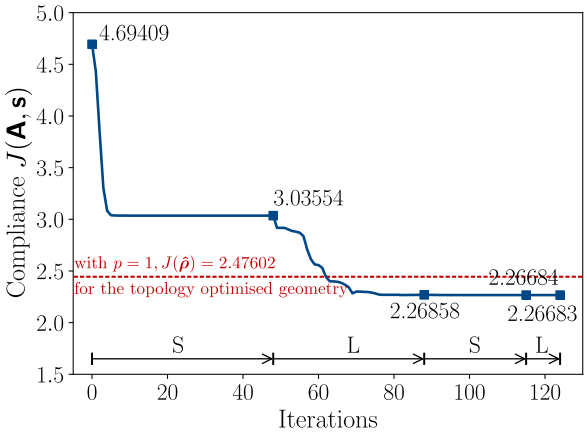
Minimum area $A_{\min}$	Maximum area $A_{\max}$	Merge ratio ζ	Tolerance ϵ_{frame}	Volume constraint
$1.5^2\pi$	$4^2\pi$	$1/20$	10^{-4}	$V \leq 28800$



(a) size optimised frame (initial)



(b) Size and layout optimised (final)



(c) Convergence

Fig. 5.11 Pipe bracket: frame size and layout optimisation. The initially size-optimised frame model (a) has 42 joints and 50 members while the final frame model (b) has 36 joints and 44 members. Red dashed line in (c) shows the structural compliance (2.47602) of the topology-optimised shape.

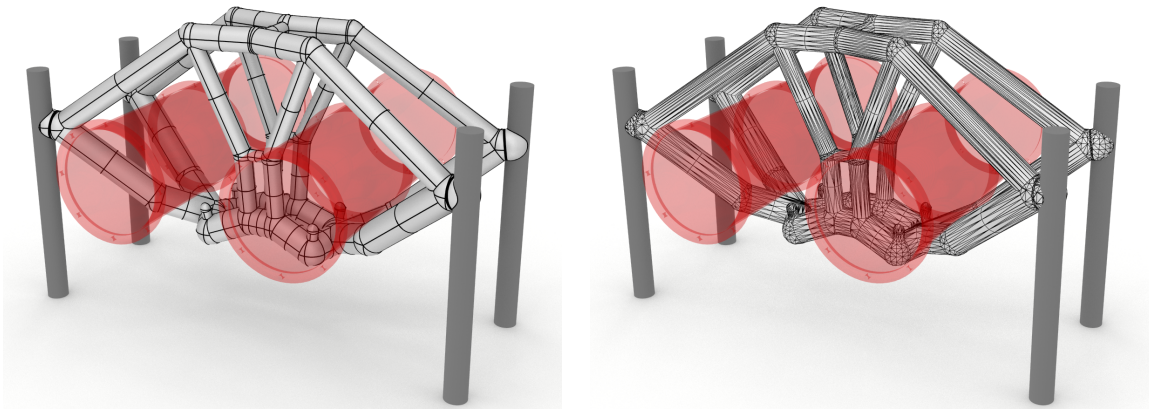
Initially all members are assumed to have the uniform cross-section areas of $A = 31.133$, giving the total volume $V = 0.1\overline{V}$. As can be seen in Figure 5.11c, the frame consisting of uniform beams has the compliance $J(\mathbf{A}, \mathbf{s}) = 4.69409$, which is higher than the topology-optimised geometry's compliance $J(\hat{\boldsymbol{\rho}}) = 2.47602$. After the first size optimisation step the compliance is reduced to $J(\mathbf{A}, \mathbf{s}) = 3.0355$ and the subsequent layout optimisation step to $J(\mathbf{A}, \mathbf{s}) = 2.26858$, which is already lower than $J(\hat{\boldsymbol{\rho}})$, see Figure 5.11c.

After two more steps, the final compliance is obtained as $J(\mathbf{A}, \mathbf{s}) = 2.26683$, which is lower than the compliance $J(\hat{\boldsymbol{\rho}}) = 2.47602$. The final optimised pipe bracket contains 36 joints and 44 members . Six edges are removed from the initial frame. Same as the in cantilever plate example, a member shorter than 1/20 of the total length of all members sharing the same node is considered as a short member, and its two end nodes are merged.

Step 5: CAD model generation

At the end of this example, Figure 5.12b shows the IGES model and a faceted triangular STL mesh of the frame structure.

The pipes that this bracket is designed to support are coloured in transparent red. With the geometric constrains embedded in the frame optimisation, the pipes fit in the optimised bracket without any intersections with the frame, which can be visually confirmed in Figure 5.12.

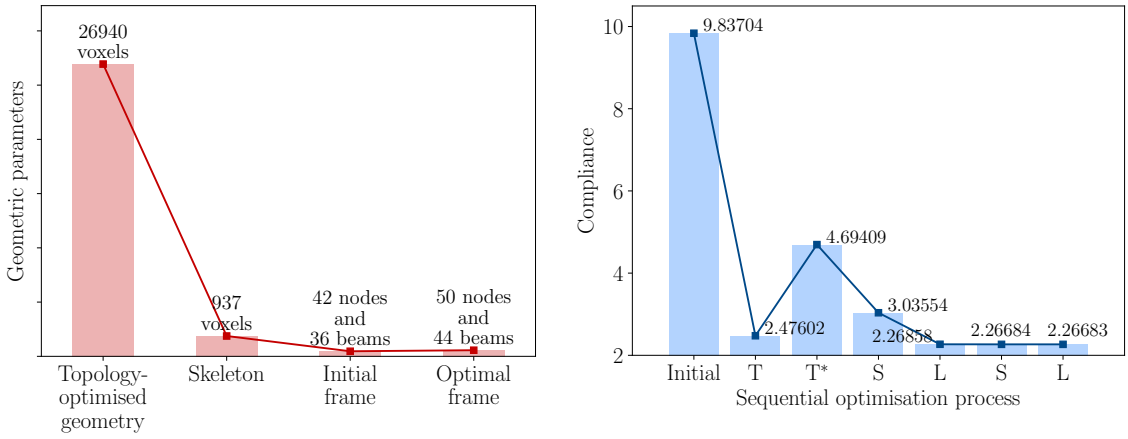


(a) IGES model (b) STL model

Fig. 5.12 Pipe bracket: parametric CAD model. IGES model (a) is the union of 36 spheres and 44 cylinders. STL mesh (b) has 8067 triangular facets.

Summary

The changes in the number of necessary geometric parameters and compliance are shown in Figure 5.13. In order to represent the final geometry, the optimal frame model uses 50 nodes and 44 beams, which is 0.35% of the number of geometric parameters used for topology-optimised model (26940 voxels). Moreover, the compliance of optimal frame has the compliance of 2.47602, which is similar to the compliance of the topology-optimised model, 2.26683.



(a) Number of geometric parameters (b) Structural compliance

Fig. 5.13 Pipe bracket: the change in the number of geometric parameters (a) and compliance (b) during the workflow. The abbreviations In (b) are Initial: the initial model with uniformly distributed density, T: topology optimisation, T*: the frame with uniform-sized beams, S: frame size optimisation and L: frame layout optimisation.

5.2.3 Rocker arm

The rocker arm is a mechanical part that is commonly found in vehicles, motorcycles and vessels. The rocker arm is designed to be lightweight hence a low volume constraint is chosen for the optimisation.

Problem description

The design domain of this rocker arm is given in Figure 5.14, which has a bounding box of size of $75 \times 60 \times 12$. The opening in the front rocker arm is left for motors and pumps. Several symmetric concentrated forces are applied to the mechanical joints, whose magnitudes and directions are displayed in Figure 5.14. The reel mechanical joints are clamped and front mechanical joints restricted not to move along the height

direction. Concentrated forces are applied on the mechanical joints on the top and the front, with the directions and magnitudes shown in Figure 5.14.

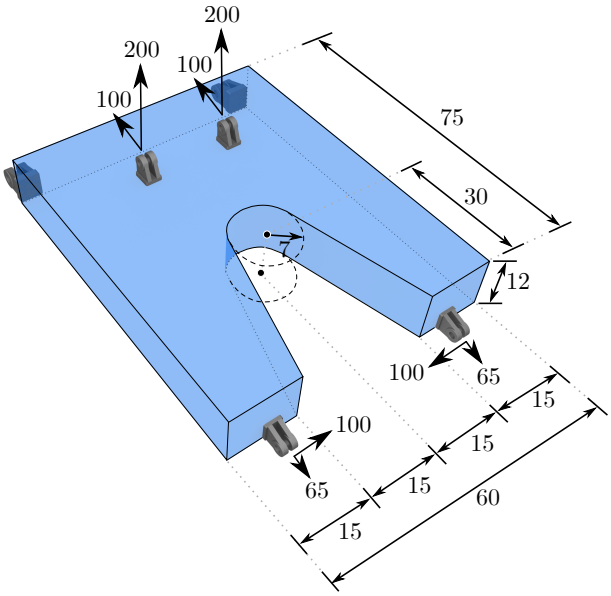


Fig. 5.14 Rocker arm: geometry, boundary conditions and loadings.

Step 1: Topology optimisation

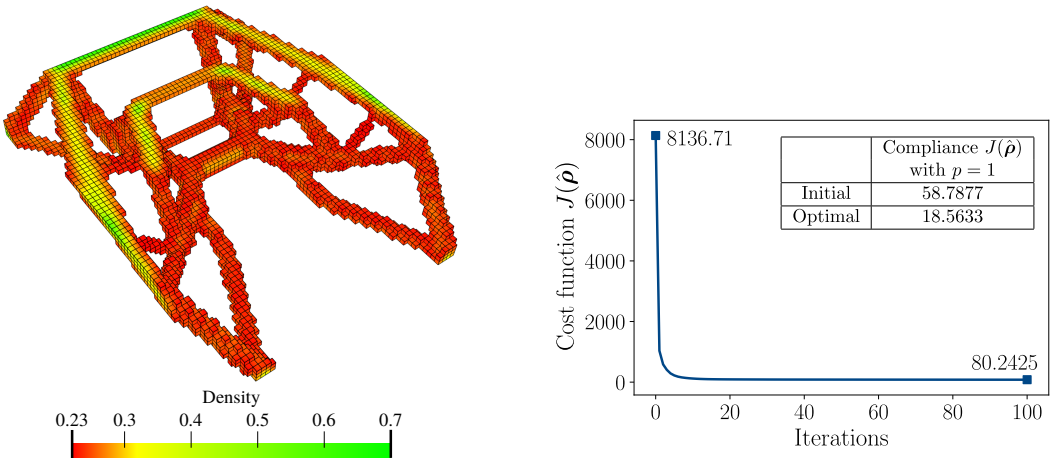
The design domain is discretised into $75 \times 60 \times 12$ linear hexahedral elements.The topology optimisation parameters are shown in Table 5.6.

Table 5.6 Rocker arm: topology optimisation parameters

Penalisation power p	Filter radius R	Volume fraction V_f	Maximum iterations
3	3	0.085	100

Since a very small volume fraction is chosen, the topology-optimised shape is with maximum density of around 0.7. This volume fraction does not allow sufficient materials to be filled in fully solid elements to present the load path. However, this is not an issue, as is visually evident in Figure 5.15a where the final shape gives a structurally sound geometry. With the initial density of 0.085, the initial cost function value is expected to be high, which is 8136.71, see Figure 5.15b. This value is reduced to 80.2425 and the optimised model is shown in Figure 5.15b. The final structural compliance is $J(\hat{\boldsymbol{\rho}}) = 18.5633$ with a reduction of 31.6% compared with the initial

compliance of 58.7877. The thresholded geometry with the threshold $\eta = 0.23$ in Figure 5.15a has the volume of 4666 $\approx V_f\overline{V} = 4590$.



(a) Thresholded geometry with $\eta = 0.23$ (b) Convergence

Fig. 5.15 Rocker arm: topology optimisation. Thresholded model (a) has 4666 voxels. The table in (b) gives the initial and final structural compliances ($J(\hat{\boldsymbol{\rho}}), p = 1$).

Step 2: Skeletonisation

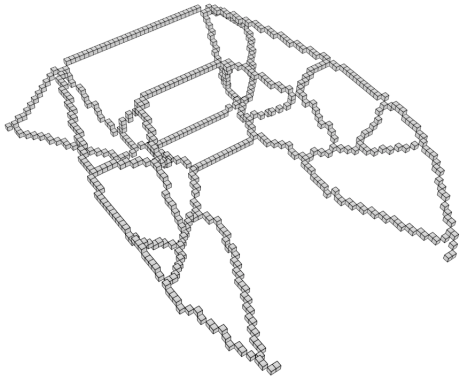


Fig. 5.16 Rocker arm: structural skeleton. The skeleton has 584 voxels.

During the skeletonisation, 3 steps are required to skeletonise the model in Figure 5.15a of 4666 voxels to a voxel chain model in Figure 5.16 of 584 voxels, with a reduction in voxel number of 87.5%.

Step 3: Frame model generation

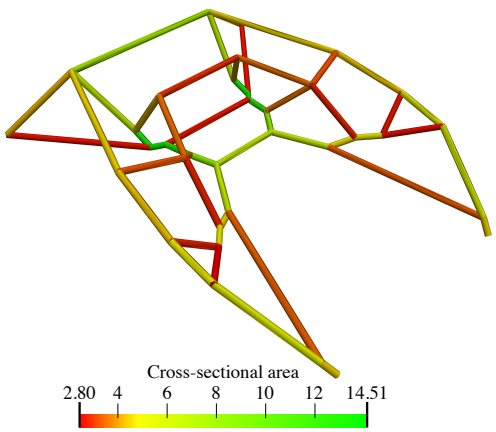
The frame in Figure 5.17a converted from the skeleton has 34 joints and 50 members.

Step 4: Frame size and layout optimisation

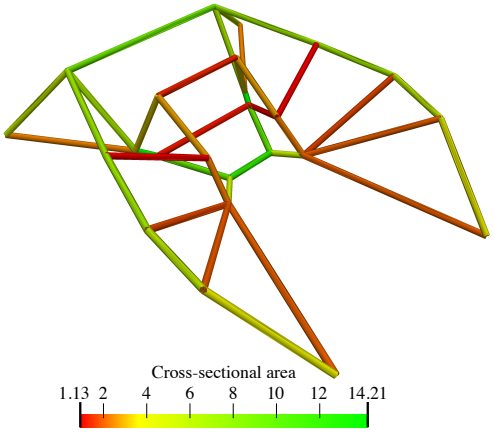
The frame optimisation has parameters given in Table 5.7.

Table 5.7 Rocker arm: frame optimisation parameters

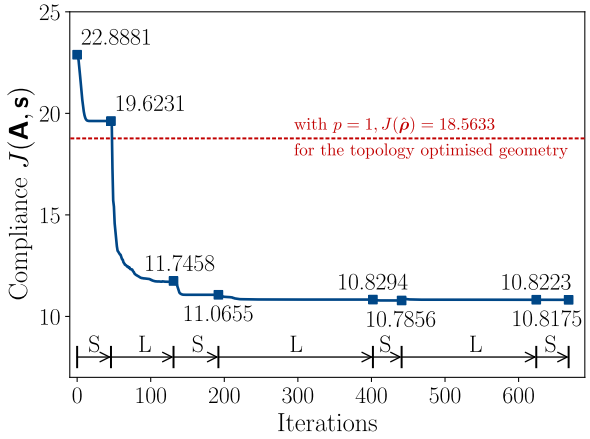
Minimum area $A_{\min}$	Maximum area $A_{\max}$	Merge ratio ζ	Tolerance ϵ_{frame}	Volume constraint
$0.6^2\pi$	–	1/20	10^{-4}	$V \leq 4590$



(a) Sized optimised frame (initial)



(b) Size and layout optimised (final)



(c) Convergence

Fig. 5.17 Rocker arm: frame size and layout optimisation. The initially size-optimised frame model (a) has 34 joints and 50 members while the final frame model (b) has 24 joints and 40 members. Red dashed line in (c) shows the structural compliance (18.5633) of the topology-optimised shape.

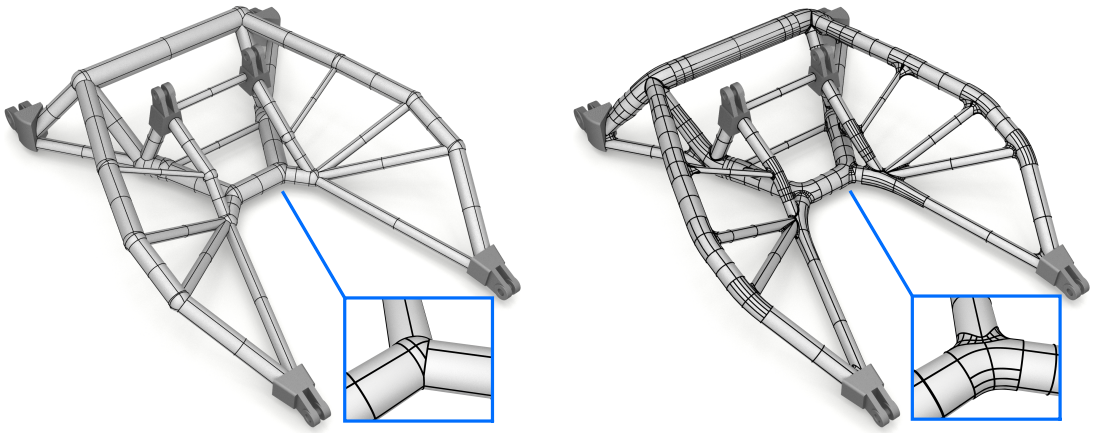
Similar to the pipe bracket example, level-set function is also used for geometric constraints here. The initial frame with volume of $V_f\overline{V} = 4590$ has uniform circular cross-sections with areas of $A = 7.8228$. The structural compliance of the initial frame is $J(\mathbf{A}, \mathbf{s}) = 22.8881 > J(\hat{\boldsymbol{\rho}}) = 18.5633$.

When sequential frame optimisation converges, the frame reaches the optimal shape having the compliance $J(\mathbf{A}, \mathbf{s}) = 10.8223 < J(\hat{\boldsymbol{\rho}}) = 18.5633$. Thus optimality has been recovered. The convergence of the compliance is shown in Figure 5.17c. The optimal frame has 24 joints and 40 members, as can be seen in Figure 5.17b; 10 members are removed during the optimisation.

Step 5: CAD model generation

As the final step of this example, CAD models are generated from the optimal frame in Figure 5.17b. Figure 5.18 shows the optimal rocker arm in IGES format and its STL mesh. As mentioned in Section 1.3, with several manual operations one can also use blend surfaces and filleted edges to generate the CAD model as shown in Figure 5.18b. The details of the trimmed curves of these two CAD models are highlighted in the blue zoom-in windows.

To smooth the edges of NURBS patches in Figure 5.18a, in Figure 5.18b, the fillet radius is set as 0.5 for simple joints that have less than 4 cylinders connected at the joint while 0.2 for relatively complicated joints that have not less than 4 cylinders connected at the joint. Note that the fillet operations cannot be completely automated, since the choice of fillet radii is entirely shape-dependant.

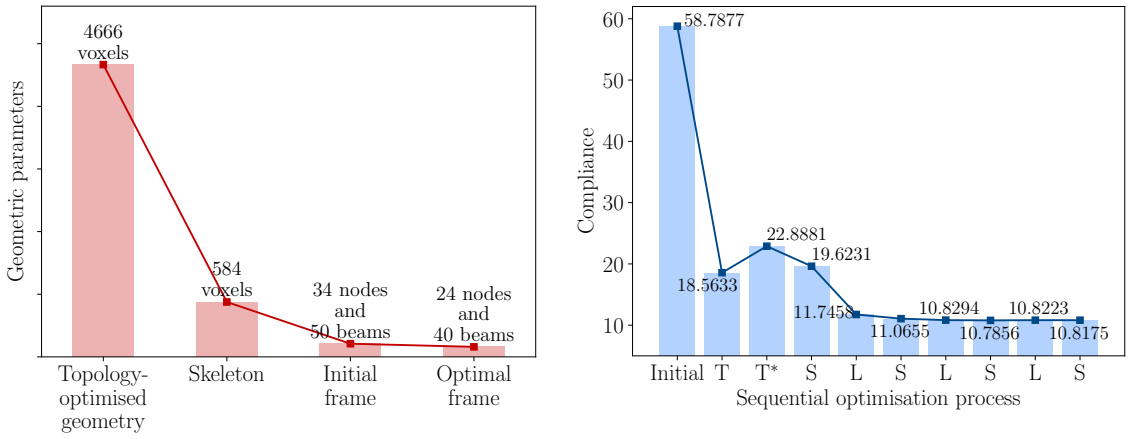


(a) IGES model (b) IGES model with filleted edges

Fig. 5.18 Rocker arm: parametric CAD models. IGES model (a) is the union of 24 spheres and 40 cylinders. (b) combines primitive solids and blended surfaces.

Summary

Again, the changes in the number of necessary geometric parameters and compliance are shown in Figure 5.19. The optimal frame model has 50 nodes and 44 beams using 1.37% of the number of geometric parameters for the topology-optimised model (4666 voxels). The compliance of optimal frame has the compliance of 18.5663 similar to the topology-optimised model (10.8175).



(a) Number of geometric parameters (b) Structural compliance

Fig. 5.19 Rocker arm: the change in the number of geometric parameters (a) and compliance (b) during the workflow. The abbreviations In (b) are Initial: the initial model with uniformly distributed density, T: topology optimisation, T*: the frame with uniform-sized beams, S: frame size optimisation and L: frame layout optimisation.

5.2.4 Frame-supported plate

In Section 5.2.2, there are two plate-like parts on the top and bottom of the optimised model in Figure 5.8a, and the skeletonisation algorithm skeletonise these two plates into chains. However, in structural design it is very common to combine a plate- or shell-like skin structure with a frame-like support structure and such plates are to be kept. Accordingly this example of a frame-supported plate is used to demonstrate how to couple the frame and plate in the optimisation process.

Problem description

The design domain shown in Figure 5.20 combines a thin plate-like skin with a solid bottom part to be optimised. The entire domain is of size $120 \times 80 \times 20$ and contains a small opening of size $40 \times 40 \times 6$. The plate is of thickness 3 and is subjected to a

uniform pressure load of 1. In such composite structures the skin can be modelled as a standard plate or shell structure, and topology optimisation is applied only in the remaining part of the design domain. The four corners of the design domain are vertically supported.

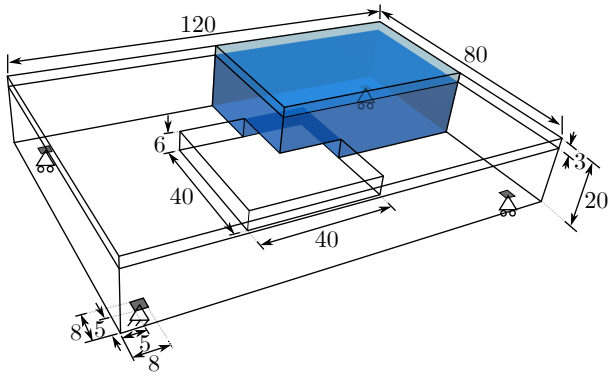


Fig. 5.20 Frame-supported plate: geometry, boundary conditions and loadings. The design domain is the entire cuboid, but in order to show the details inside, only a quarter of the domain is coloured in blue.

Step 1: Topology optimisation

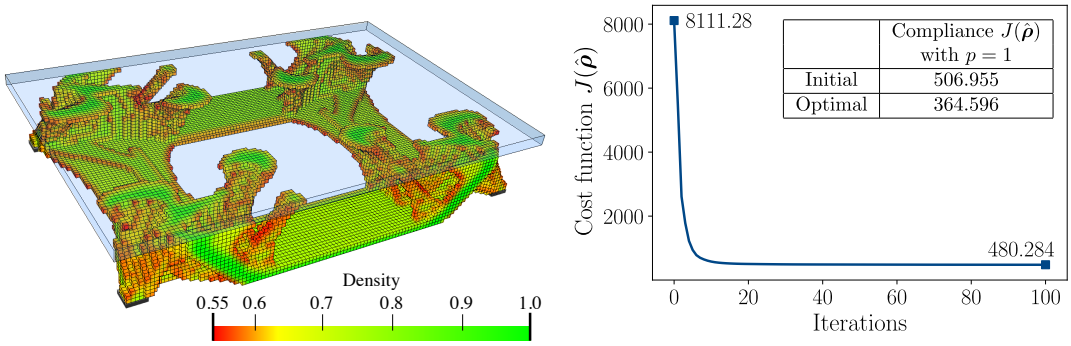
The finite element discretisation consists of $120 \times 80 \times 23$ linear hexahedral finite elements. Across the height of the design domain there are 6 elements of size $1 \times 1 \times 0.5$ in the plate-like top part and 17 unit elements in the bottom part. A lower thickness is chosen for the elements of the plate to avoid the plate to be unnecessarily stiff to avoid the *shear locking*[1] caused by using thick elements in modelling a thin plate [166]. Topology optimisations are conducted using even smaller thicknesses for the top plate elements, however there are negligible differences in initial and final cost function values. The topology optimisation parameters are shown in Table 5.8. Note that the density of these elements are fixed at 1 during the topology optimisation.

The maximum volume fraction is prescribed to be $V_f = 0.25$. The plate on the top with volume of $0.15\overline{V} = 28800$ is set to be always solid. Therefore the optimisation on the frame below the plate has the volume constraint of $0.1\overline{V} = 19200$.

[1]Shear locking is an error that occurs in finite element analysis due to the linear nature of 2D/3D elements. The linear elements do not accurately model the curvature present in the actual material under bending, and a shear stress is introduced. The additional shear stress in the element causes the element to reach equilibrium with smaller displacements, i.e., it makes the element appear to be stiffer than it actually is and gives bending displacements smaller than they should be.

Table 5.8 Frame-supported plate: topology optimisation parameters

Penalisation power p	Filter radius R	Volume fraction V_f	Maximum iterations
3	1.5	0.25	100



(a) Thresholded geometry with $\eta = 0.55$ (b) Convergence

Fig. 5.21 Frame-supported plate: topology optimisation. Thresholded model (a) has 22704 voxels and the plate is hidden for clarity. The table in (b) gives the initial and final structural compliances ($J(\hat{\boldsymbol{\rho}}), p = 1$).

During the topology optimisation the cost function value reduces from 8111.28 of the initial shape to 478.320. The optimal structural compliance is 364.596. The optimised structure with only the voxels above a relative density $\eta = 0.55$ is shown in Figure 5.21a which gives a thresholded model including frame part plus the top plate with the volume of $22704 \approx 19200 = 0.1\overline{V}$.

Step 2: Skeletonisation

Before the skeletonisation, two things need to be clarified. (i) The voxels in the top part of the design domain are tagged as non-removable to keep the skin of this composite structure. (ii) Though the elements of the top plate have a different size from that of the voxels on the bottom frame, the mesh of the design domain is still structured; so all voxels are treated of the same size when determining their 26-neighbourhoods. In doing so, the checks for simple points can still use the definition provided in Section 3.2.2. For unstructured mesh, the definition of simple points should be referred to [91].

The skeletonisation algorithm uses 16 steps to skeletonise the voxel model of 22856 voxels into a voxel chain of 976 voxels.

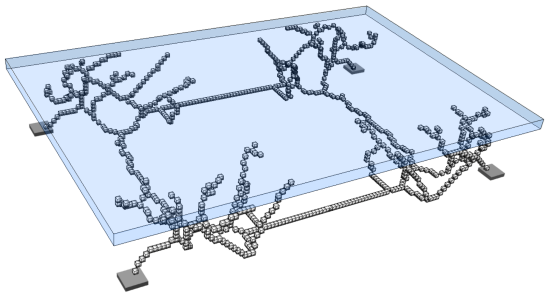


Fig. 5.22 Frame-supported plate: structural skeleton. The skeleton has 976 voxels.

Step 3: Frame model generation

The voxel chain skeleton is converted to the frame structure shown in Figure 5.23a. The frame has 124 joints and 136 members.

Step 4: Frame size and layout optimisation

The structural model also contains a top plate not shown in Figure 5.23. The plate is modelled as a Kirchhoff–Love plate and is discretised with 4×4 quadrilateral elements and Catmull-Clark subdivision basis functions. The finite element analysis conducted here couples the shell and frame. The coupling between the plate and the frame is achieved with Lagrange multipliers as described in [167]. The joints between the plate and frame can transfer forces but not moments and the positions of the joints between the frame and plate are fixed during the optimisation.

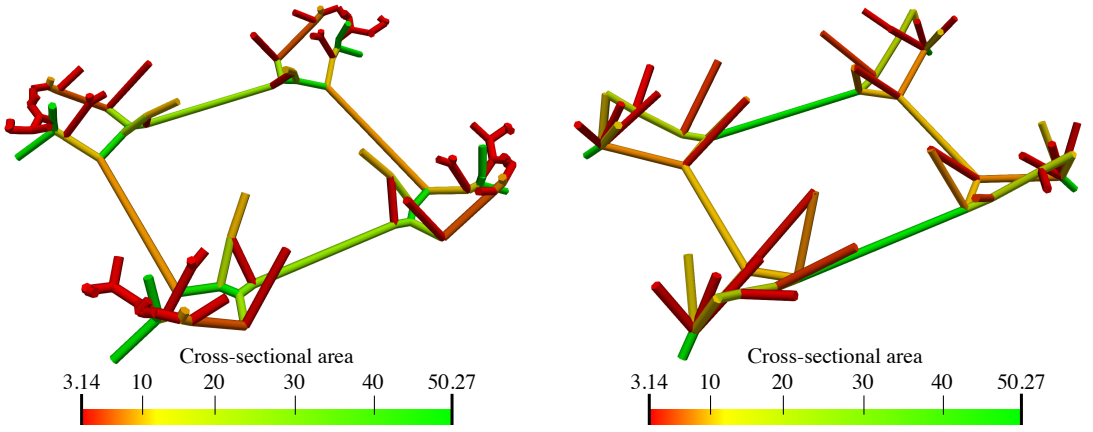
The frame optimisation uses parameters in Table 5.9. Initially all the members are assumed to have the same cross-section area of $A = 19.710$ giving the total frame volume of $V = 19200$. According to Figure 5.23c, the compliance of the frame-supported plate of members with uniform cross-section areas is $J(\mathbf{A}, \mathbf{s}) = 1210.01$. After the first step of size and layout optimisation, the structural compliance reduces to 391.283, which is close to the compliance of the topology-optimised model. Compliance keeps decreasing during the sequential size and layout optimisation steps and converges at

Table 5.9 Frame-supported plate: frame optimisation parameters

Minimum area $A_{\min}$	Maximum area $A_{\max}$	Merge ratio ζ	Tolerance ϵ_{frame}	Volume constraint
$1^2\pi$	$4^2\pi$	$1/20$	10^{-4}	$V \leq 19200$

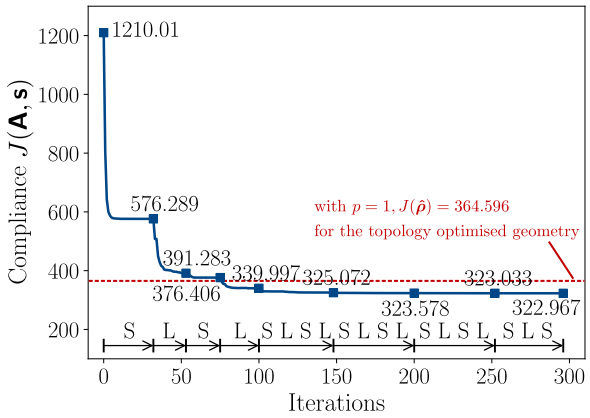
$J(\mathbf{A}, \mathbf{s}) = 322.967 < J(\hat{\boldsymbol{\rho}}) = 364.596$. Thus, the optimality has been recovered. The final optimised frame structure contains 56 joints and 64 members, see Figure 5.23b.

The increase in stiffness due to optimisation is also evident from the deflected shapes shown in Figure 5.24. The size and layout optimisations make the top plate deform less, especially the displacements on the parts without frames to support, e.g. the mid-span of the shell. In this thesis, the connecting joints of the frame to the shell are fixed. Thus it is expected that the compliance of the frame can decrease even further if the coordinates of these connecting joints are optimised. Unfortunately, relevant optimisation techniques for connecting joints have not been explored by researchers. Since the optimal frame provides a structural performance similar to that of the



(a) Sized optimised frame (initial)

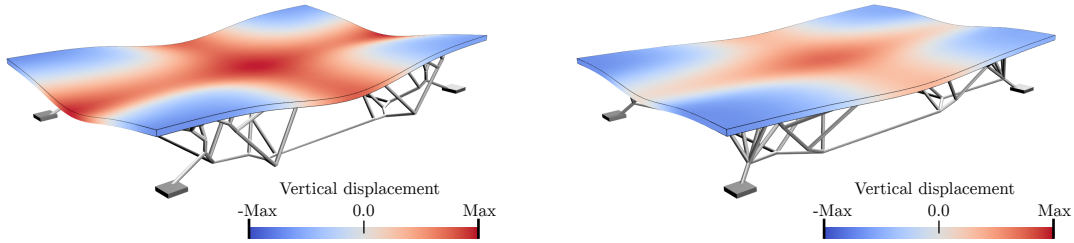
(b) Size and layout optimised frame (final)



(c) Convergence

Fig. 5.23 Frame-supported plate: frame size and layout optimisation. The initially size-optimised frame model (a) has 124 joints and 136 members while the final frame model (b) has 56 joints and 64 members. Red dashed line in (c) shows the structural compliance (364.596) of the topology-optimised shape.

topology-optimised geometry, the positions of connecting joints are considered to be fixed at the current stage.



(a) The frame below the plate is size optimised (initial)

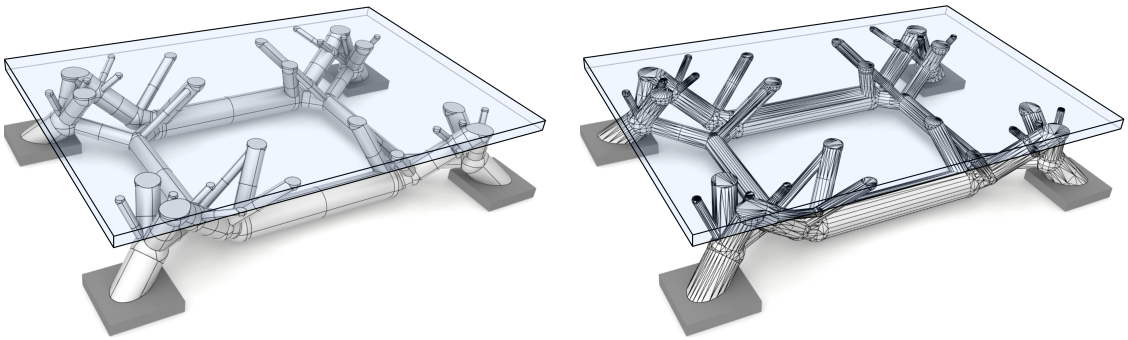
(b) The frame below the plate is size and layout optimised (final)

Fig. 5.24 Frame-supported plate: deformation distribution of the top plate. The value of deformation from lowest to highest is coloured from blue to red. The displacement distribution in (b) is lower than the on in (a), which gives an intuitive of the increase of the stiffness of the plate after size and layout optimisation.

The optimal frame has 56 joints and 64 members. 72 members are deleted from the initial frame.

Step 5: CAD model generation

The solid CAD model of this frame-supported plate and its faceted triangular STL mesh are shown in Figure 5.25. The top plate is transparent for clarity.



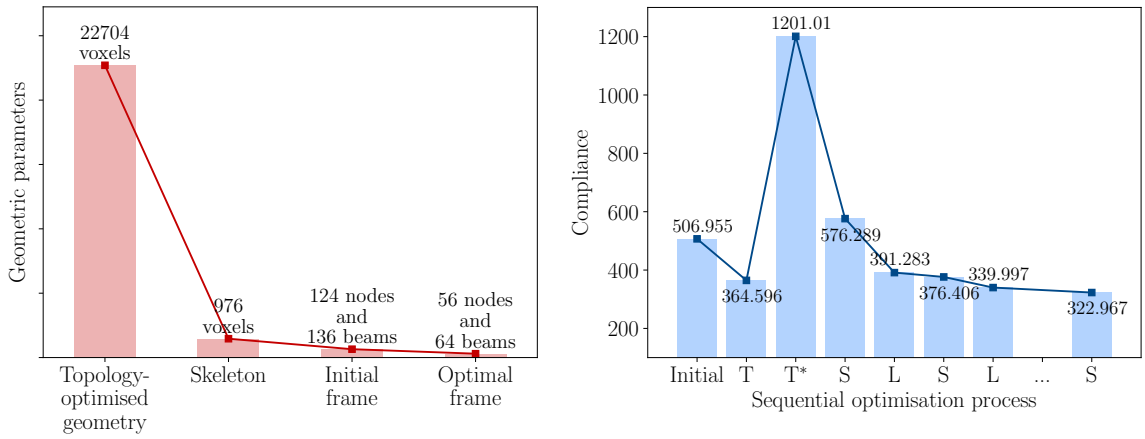
(a) IGES model

(b) STL model

Fig. 5.25 Frame-supported plate: parametric CAD model. IGES model (a) is the union of 56 spheres, 64 cylinders and 1 plate. STL mesh (b) has 11154 triangular facets.

Summary

In the case where frame and shell are coupled, the proposed workflow can simplify the frame-like structure beneath the plate as well as preserve its optimality. As shown in Figure 5.26a, the complexity of the geometry drops from 22704 voxels (topology-optimised model) to 56 nodes plus 64 beams (final optimal frame). In Figure 5.26b, the final model has the compliance of 322.967 which is modelled using the optimal frame and the thin-shell. This is close to the solid topology-optimised model with the compliance of 364.596. Thus, optimality found by topology optimisation has been preserved.



(a) Number of geometric parameters (b) Structural compliance

Fig. 5.26 Frame-supported plate: the change in the number of geometric parameters (a) and compliance (b) during the workflow. The abbreviations In (b) are Initial: the initial model with uniformly distributed density, T: topology optimisation, T*: the frame with uniform-sized beams, S: frame size optimisation and L: frame layout optimisation.

CHAPTER 6

Conclusions and future work

6.1 Conclusions

The automated and optimality-preserving conversion of topology-optimised geometries to compact parametric CAD models is a missing link in the wide-scale industrial adoption of optimisation. This thesis introduces a fully automated workflow to synthesise a structurally sound parametric CAD model from a voxelised solid-void binary image obtained with topology optimisation. Currently, there is no commercial design system, such as Altair Inspire [168], Abaqus CAE [169] or similar, which offers such a fully automated CAD model generation capability.

The performed research and the achievements of this thesis are as follows.

- Homotopic skeletonisation extracts the curve skeleton from the topology optimised model. In this thesis, the skeletonisation of three-dimensional images is a key component of the proposed workflow, which is an extensively studied topic in digital topology. The solid mathematical foundation of the skeletonisation process ensures its robustness. This thesis introduces the concept of tagging voxels to preserve important structural and geometric features, and extend the skeletonisation algorithm in industrial design. The topology-preserving and feature-preserving skeletonisation provides a faithful load path of the structural-optimised structure.
- Graph algorithms edit the curve skeleton. The resulting voxel chain skeleton is interpreted as a weighted undirected graph and processed with the proposed graph algorithms to yield a structurally sound frame model. The ill-performing

frame members such as short beams and zero-stress members can easily be detected and deleted using the proposed graph algorithms in this thesis. The graph algorithms operate on topology rather than geometry and are not subject to floating-point errors, which makes them exceedingly robust and efficient.

- Frame optimisation recovers optimality. The obtained structural frame models are not optimal because the skeletonisation and graph algorithms do not take into account any mechanical consideration. However, as shown in the numerical examples in Chapter 5, after several alternating steps of size and layout optimisation, the structural frames can achieve a structural compliance similar to that of the original topology-optimised geometry. Moreover, with the implicitly represented geometric constraints introduced in this thesis, the frame optimisation can proceed within complex design domains, which are commonly seen in industrial design.

- CSG tree constructs compact CAD solid models. The frame model is converted into a compact CAD solid model by recursively combining primitive shapes using boolean operations. Although in this thesis only cylinders and spheres are used as shape primitives, it is straightforward to use other primitives.

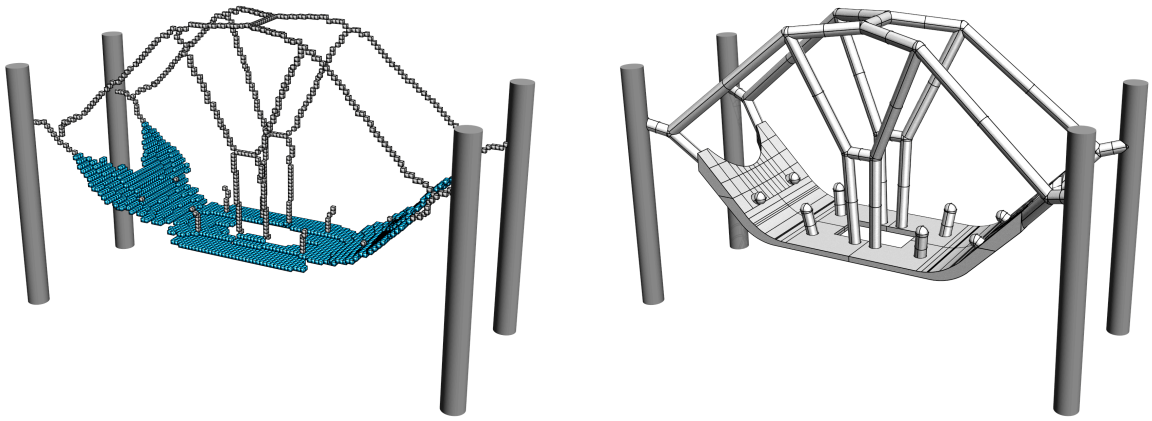
The generated parametric CAD solid model and its compact CSG tree representation can be edited to refine the optimised design or to combine it with other components in a product. It is worth reminding that in industrial practice, as supported by user studies [170], geometries obtained from optimisation are often the starting point for design exploration and not the endpoint. This usually requires the ability to edit the optimised geometries in a CAD system. A high-level compact representation is also useful in taking into account geometric manufacturing constraints, like minimum thickness, slope or length of the members [171]. Beyond geometry editing, the frame model may also have advantages in structural analysis of the CAD model. The frame model makes it, for instance, easier to check the structure for buckling and inelastic deformations. As a low-order model, it can be analysed orders of magnitude faster than a three-dimensional solid model. This opens up the possibility of instant analysis of CAD models as it has become recently available in, e.g., Ansys Discovery Live [172] or Creo Simulation Live [173].

6.2 Future work

In closing, the direct extensions to the proposed CAD generation process are listed as follows:

- In the proposed approach, it is assumed that the topology-optimised geometry can be approximated with tubular members. However, in three-dimensional optimisation problems, large flat or curved surfaces may also appear during optimisation. The skeletonisation algorithm introduced in Section 3.2 will reduce even those to a network of one voxel thick chains. The surface skeletonisation discussed in Section 3.2.2 preserves one-voxel thick surfaces. In Figure 6.1 the topology-optimised pipe bracket has been skeletonised with the surface skeletonisation algorithm. The curved surface at the bottom is preserved. In the structural model, this surface can be modelled as a thin-shell and in the CAD model it can be easily converted to a solid by providing a thickness. Currently, there is no automated process to extract a shell surface from the skeletonised voxel model, but this seems to be possible and is a promising direction for future research.

- Straight frame members are used in the workflow in this thesis. This has the assumption that loadings are applied on the frame nodes/joints. Since approaches with the concept of *isogemetric analysis* flourishing these days, CAD compatible geometry representations, such as B-spline or NURBS, can also be used to model



(a) Voxel chain skeleton from surface-preserving skeletonisation

(b) IGES model with bottom plate constructed based on the blue voxels in (a)

Fig. 6.1 CAD model generated from surface skeleton. The topology-optimised pipe bracket comes from the voxel model in Figure 5.8a.

the frame members in finite element. This needs the frame optimisation to be adjusted to B-spline or NURBS basis functions.

- In the frame-supported plate example in Section 5.2.4, the thickness and shape of the plate modelled as a thin-shell were fixed and only the frame beneath the plate was optimised. However, it is promising that coupling shell and frame optimisation in the proposed approach can provide an automated, optimality-preserving CAD generation workflow for lattice structures and composite structures.

- From a optimal frame model, one can also generate a structured surface mesh, as mentioned in Section 4.2.1. In terms of the shape, this surface mesh is similar to a hollowed topology optimised model. Moreover, this mesh is coarse and represents all essential topological features of the topology-optimised geometry. In the case where shell structures are desired, one can conduct shape optimisation on this mesh using existing techniques, e.g. [13]. With the booming knowledge of mesh and CAD model reconstruction, more immediate future work can be carried on based on the optimal frame model.

- The workflow can be further developed to take the input from the topology optimisation with multi-phase materials. This is to help the design of multi-phase material products. In this case, one possible way is to first segment the topology-optimised volume mesh according to different material settings. Then the skeletonisation algorithm is applied on each segmentation and these skeletons are connected at the borders of segmentations. After converting the skeleton model in to frame models, different materials are assigned to the corresponding frame members. The rest of the workflow are identical to the one described in this thesis.

References

[1] Martti Mantyla. *Introduction to solid modeling*. WH Freeman & Co., 1988.

[2] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

[3] James D Foley, Foley Dan Van, Andries Van Dam, Steven K Feiner, John F Hughes, Edward Angel, and J Hughes. *Computer graphics: principles and practice*, volume 12110. Addison-Wesley Professional, 1996.

[4] Peter W Christensen and Anders Klarbring. *An introduction to structural optimization*, volume 153. Springer Science & Business Media, 2008.

[5] Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39-41):4135–4195, 2005.

[6] Vincent Braibant and Claude Fleury. Shape optimal design using b-splines. *Computer Methods in Applied Mechanics and Engineering*, 44(3):247–267, 1984.

[7] K-U Bletzinger, Stefan Kimmich, and Ekkehard Ramm. Efficient modeling in shape optimal design. *Computing Systems in Engineering*, 2(5-6):483–495, 1991.

[8] Trevor T Robinson, Cecil G Armstrong, Hung Soon Chua, Carsten Othmer, and Thorsten Grahs. Optimizing parameterized cad geometries using sensitivities based on adjoint functions. *Computer-Aided Design and Applications*, 9(3):253–268, 2012.

[9] Xiaocong Han and David W Zingg. An adaptive geometry parametrization for aerodynamic shape optimization. *Optimization and Engineering*, 15(1):69–91, 2014.

[10] Fehmi Cirak, Michael J Scott, Erik K Antonsson, Michael Ortiz, and Peter Schröder. Integrated modeling, finite-element analysis, and engineering design for thin-shell structures using subdivision. *Computer-Aided Design*, 34(2):137–148, 2002.

[11] Wolfgang A Wall, Moritz A Frenzel, and Christian Cyron. Isogeometric structural shape optimization. *Computer methods in applied mechanics and engineering*, 197(33-40):2976–2988, 2008.

[12] Josef Kiendl, K-U Bletzinger, Johannes Linhard, and Roland Wüchner. Isogeometric shell analysis with kirchhoff–love elements. *Computer Methods in Applied Mechanics and Engineering*, 198(49-52):3902–3914, 2009.

[13] Kosala Bandara, Thomas Rüberg, and Fehmi Cirak. Shape optimisation with multiresolution subdivision surfaces and immersed finite elements. *Computer Methods in Applied Mechanics and Engineering*, 300:510–539, 2016.

[14] Austin J Herrema, Nelson M Wiese, Carolyn N Darling, Baskar Ganapathysubramanian, Adarsh Krishnamurthy, and Ming-Chen Hsu. A framework for parametric design optimization using isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:944–965, 2017.

[15] Kosala Bandara and Fehmi Cirak. Isogeometric shape optimisation of shell structures using multiresolution subdivision surfaces. *Computer-Aided Design*, 95:62–71, 2018.

[16] Wolfgang A Wall, Moritz A Frenzel, and Christian Cyron. Isogeometric structural shape optimization. *Computer methods in applied mechanics and engineering*, 197(33):2976–2988, 2008.

[17] Seonho Cho and Seung-Hyun Ha. Isogeometric shape design optimization: exact geometry and enhanced sensitivity. *Structural and Multidisciplinary Optimization*, 38(1):53, 2009.

[18] Yu-Deok Seo, Hyun-Jung Kim, and Sung-Kie Youn. Isogeometric topology optimization using trimmed spline surfaces. *Computer Methods in Applied Mechanics and Engineering*, 199(49-52):3270–3296, 2010.

[19] Ole Sigmund and Kurt Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055, 2013.

[20] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184. ACM, 2004.

[21] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[22] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.

[23] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics interface*, volume 98, pages 43–50, 1998.

[24] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[25] Francesco Buonamici, Monica Carfagni, Rocco Furferi, Lapo Governi, Alessandro Lapini, and Yary Volpe. Reverse engineering modeling methods and tools: a survey. *Computer-Aided Design and Applications*, 15(3):443–464, 2018.

[26] Alan P Mangan and Ross T Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[27] Michael Garland, Andrew Willmott, and Paul S Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58. ACM, 2001.

[28] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 905–914. ACM, 2004.

[29] Dong-Ming Yan, Wenping Wang, Yang Liu, and Zhouwang Yang. Variational mesh segmentation via quadric surface fitting. *Computer-Aided Design*, 44(11):1072–1082, 2012.

[30] Matthias Eck and Hugues Hoppe. *Automatic reconstruction of B-spline surfaces of arbitrary topological type*. Technische Hochschule Darmstadt. Fachbereich Mathematik, 1996.

[31] Mohammad Rouhani, Angel D Sappa, and Edmond Boyer. Implicit b-spline surface reconstruction. *IEEE Transactions on Image Processing*, 24(1):22–32, 2014.

[32] Martin Marinov, Marco Amagliani, Tristan Barback, Jean Flower, Stephen Barley, Suguru Furuta, Peter Charrot, Iain Henley, Nanda Santhanam, G Thomas Finnigan, et al. Generative design conversion to editable and watertight boundary representation. *Computer-Aided Design*, 115:194–205, 2019.

[33] K Maute and E Ramm. Adaptive topology optimization. *Structural optimization*, 10(2):100–112, 1995.

[34] C-Y Lin and L-S Chao. Automated image interpretation for integrated topology and shape optimization. *Structural and Multidisciplinary Optimization*, 20(2):125–137, 2000.

[35] Yeh-Liang Hsu, Ming-Sho Hsu, and Chuan-Tang Chen. Interpreting results from topology optimization using density contours. *Computers & Structures*, 79:1049–1058, 2001.

[36] Louisa Lam, Seong-Whan Lee, and Ching Y Suen. Thinning methodologies-a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 14:869–885, 1992.

[37] Nicu D Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization & Computer Graphics*, (3):530–548, 2007.

[38] Punam K Saha, Gunilla Borgefors, and Gabriella Sanniti di Baja. A survey on skeletonization algorithms and their applications. *Pattern Recognition Letters*, 76:3–12, 2016.

[39] Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. In *Computer Graphics Forum*, volume 35, pages 573–597. Wiley Online Library, 2016.

[40] Damian J Sheehy, Cecil G Armstrong, and Desmond J Robinson. Shape description by medial surface construction. *IEEE Transactions on Visualization and Computer Graphics*, 2:62–72, 1996.

[41] Michael Bremicker, Mehran Chirehdast, Noboru Kikuchi, and PY Papalambros. Integrated topology and shape optimization in structural design. *Journal of Structural Mechanics*, 19(4):551–587, 1991.

[42] Carlo Arcelli, Luigi P Cordella, and Stefano Levialdi. From local maxima to connected skeletons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):134–143, 1981.

[43] Ta-Chih Lee, Chong-Nam Chu, and Rangasami L. Kashyap. Cad/cam integration via skeleton-based modeling. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, pages 7–12. IEEE, 1992.

[44] Ta-Chih Lee, Rangasami L Kashyap, and Chong-Nam Chu. Building skeleton models via 3-d medial surface axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56:462–478, 1994.

[45] Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2009.

[46] T Yung Kong and Azriel Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, 1989.

[47] Sergio Pellegrino. Structural computations with the singular value decomposition of the equilibrium matrix. *International Journal of Solids and Structures*, 30(21):3025–3035, 1993.

[48] Jean-Christophe Cuillière, Vincent François, and Alexandre Nana. Automatic construction of structural cad models from 3d topology optimization. *Computer-Aided Design and Applications*, 15(1):107–121, 2018.

[49] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Transactions on Graphics (TOG)*, 27(3):44, 2008.

[50] Jean-Christophe Cuillière and Vincent Francois. Integration of cad, fea and topology optimization through a unified topological model. *Computer-aided design and applications*, 11(5):493–508, 2014.

[51] Ge Yin, Xiao Xiao, and Fehmi Cirak. Topologically robust CAD model generation for structural optimisation. https://arxiv.org/abs/1906.07631v1.

[52] Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 192:227–246, 2003.

[53] Grégoire Allaire, François Jouve, and Anca-Maria Toader. Structural optimization using sensitivity analysis and a level-set method. *Journal of computational physics*, 194(1):363–393, 2004.

[54] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71:197–224, 1988.

[55] Martin Philip Bendsøe. Optimal shape design as a material distribution problem. *Structural Optimization*, 1:193–202, 1989.

[56] Martin P Bendsøe and Ole Sigmund. *Topology optimization: theory, methods and applications*. Springer, 2003.

[57] George IN Rozvany. Historical review of layout optimization. In *Structural optimization, proceedings of the IUTAM symposium on structural optimization, Melbourne*, 1988.

[58] Sarah Louise Mitchell. *Topology optimization of silicon anode structures for lithium-ion battery applications*. PhD thesis, California Institute of Technology, 2016.

[59] Ole Sigmund. On the design of compliant mechanisms using topology optimization. *Journal of Structural Mechanics*, 25(4):493–524, 1997.

[60] Hans A Eschenauer and Niels Olhoff. Topology optimization of continuum structures: a review. *Applied Mechanics Reviews*, 54(4):331–390, 2001.

[61] Harald Fredricson. Structural topology optimisation: an application review. *International journal of vehicle design*, 37(1):67–80, 2005.

[62] Fengwen Wang, Boyan Stefanov Lazarov, and Ole Sigmund. On projection methods, convergence and robust formulations in topology optimization. *Structural and Multidisciplinary Optimization*, 43(6):767–784, 2011.

[63] T Dbouk. A review about the engineering design of optimal heat transfer systems using topology optimization. *Applied Thermal Engineering*, 112:841–854, 2017.

[64] RJ Yang, Ching-Hung Chuang, Xiangdong Che, and Ciro Soto. New applications of topology optimisation in automotive industry. *International Journal of Vehicle Design*, 23(1-2):1–15, 2000.

[65] George IN Rozvany. A critical review of established methods of structural topology optimization. *Structural and multidisciplinary optimization*, 37(3):217–237, 2009.

[66] Katsuyuki Suzuki and Noboru Kikuchi. A homogenization method for shape and topology optimization. *Computer methods in applied mechanics and engineering*, 93(3):291–318, 1991.

[67] José Miranda Guedes and Noboru Kikuchi. Preprocessing and postprocessing for materials based on the homogenization method with adaptive finite element methods. *Computer methods in applied mechanics and engineering*, 83(2):143–198, 1990.

[68] Behrooz Hassani. Homogenization and structural topology optimization. 275(37):28882–28887, 1999.

[69] Grégoire Allaire, François Jouve, and Anca-Maria Toader. A level-set method for shape optimization. *Comptes Rendus Mathematique*, 334(12):1125–1130, 2002.

[70] Ming Zhou and George IN Rozvany. The COC algorithm, part ii: Topological, geometrical and generalized shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 89(1-3):309–336, 1991.

[71] Chandrashekhar S Jog and Robert B Haber. Stability of finite element models for distributed-parameter optimization and topology design. *Computer methods in applied mechanics and engineering*, 130(3-4):203–226, 1996.

[72] Alejandro Diaz and Ole Sigmund. Checkerboard patterns in layout optimization. *Structural optimization*, 10(1):40–45, 1995.

[73] Ole Sigmund and Joakim Petersson. Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural and Multidisciplinary Optimization*, 16(1):68–75, 1998.

[74] Blaise Bourdin. Filters in topology optimization. *International journal for numerical methods in engineering*, 50(9):2143–2158, 2001.

[75] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.

[76] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.

[77] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

[78] Susana Rojas-Labanda and Mathias Stolpe. Benchmarking optimization solvers for structural topology optimization. *Structural and Multidisciplinary Optimization*, 52(3):527–547, 2015.

[79] Ivo Babuška. The finite element method with lagrangian multipliers. *Numerische Mathematik*, 20(3):179–192, 1973.

[80] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[81] Yun Kang Sui and Xi Cheng Wang. Second-order method of generalized geometric programming for spatial frame optimization. *Computer methods in applied mechanics and engineering*, 141(1-2):117–123, 1997.

[82] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Optimization and engineering*, 8(1):67, 2007.

[83] Yun-kang Sui. The expansion of functions under transformation and its application to optimization. *Computer Methods in Applied Mechanics and Engineering*, 113(3-4):253–262, 1994.

[84] Gerhard Kreisselmeier and R Steinhauser. Systematic control design by optimizing a vector performance index. In *Computer aided design of control systems*, pages 113–117. Elsevier, 1980.

[85] JRRA Martins and Nicholas MK Poon. On structural optimization using constraint aggregation. In *VI World Congress on Structural and Multidisciplinary Optimization WCSMO6, Rio de Janeiro, Brasil*, 2005.

[86] Makoto Ohsaki. *Optimization of finite dimensional structures*. CRC Press, 2016.

[87] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999.

[88] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. volume 32, page 27. ACM, 2013.

[89] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. Openvdb: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*, page 19. ACM, 2013.

[90] Azriel Rosenfeld. A characterization of parallel thinning algorithms. *Information and control*, 29(3):286–291, 1975.

[91] Ying Bai, Xiao Han, and Jerry L Prince. Digital topology on adaptive octree grids. *Journal of mathematical imaging and vision*, 34(2):165–184, 2009.

[92] Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

[93] Harry Blum et al. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967.

[94] Fernand Meyer. *Cytologie quantitative et morphologie mathématiques*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris (Université de soutenance), Paris, France, 1979.

[95] Céline Fouard, E Cassot, G Malandain, C Mazel, S Prohaska, D Asselot, M Westerhoff, and Jean-Pierre Marc-Vergnes. Skeletonization by blocks for large 3d datasets: application to brain microcirculation. In *Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on*, pages 89–92. IEEE, 2004.

[96] Michel Couprie. Note on fifteen 2d parallel thinning algorithms. *Université de Marne-la-Vallée, rap. tech*, 2006.

[97] R Ogniewicz and Markus Ilg. Voronoi skeletons: Theory and applications. In *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 63–69. IEEE, 1992.

[98] Avneesh Sud. *Efficient computation of discrete Voronoi diagram and homotopy-preserving simplified medial axis of a 3D polyhedron*. PhD thesis, University of North Carolina at Chapel Hill, 2006.

[99] Jonathan W Brandt and V Ralph Algazi. Continuous skeleton computation by voronoi diagram. *CVGIP: Image understanding*, 55(3):329–338, 1992.

[100] Michal Etzion and Ari Rappoport. Computing voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry*, 21(3):87–120, 2002.

[101] Tamal K Dey and Wulue Zhao. Approximate medial axis as a voronoi subcomplex. *Computer-Aided Design*, 36(2):195–202, 2004.

[102] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry*, 19(2-3):127–153, 2001.

[103] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266. ACM, 2001.

[104] Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 35–42. ACM, 2001.

[105] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 954–961. ACM, 2003.

[106] Michela Mortara, Giuseppe Patanè, Michela Spagnuolo, Bianca Falcidieno, and Jarek Rossignac. Plumber: a method for a multi-scale decomposition of 3d shapes into tubular primitives and bodies. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 339–344. Eurographics Association, 2004.

[107] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8-10):649–658, 2005.

[108] Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patané, Michela Spagnuolo, and Ayellet Tal. Mesh segmentation-a comparative study. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, pages 7–7. IEEE, 2006.

[109] Andrei Sharf, Thomas Lewiner, Ariel Shamir, and Leif Kobbelt. On-the-fly curve-skeleton computation for 3d shapes. In *Computer Graphics Forum*, volume 26, pages 323–328. Wiley Online Library, 2007.

[110] Andrei C Jalba, Jacek Kustra, and Alexandru C Telea. Surface and curve skeletonization of large 3d models on the gpu. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1495–1508, 2013.

[111] Alexander Bucksch and Roderik Lindenbergh. Campino—a skeletonization method for point cloud processing. *ISPRS journal of photogrammetry and remote sensing*, 63(1):115–127, 2008.

[112] Jaehwan Ma, Sang Won Bae, and Sunghee Choi. 3d medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28(1):7–19, 2012.

[113] Jacek Kustra, Andrei Jalba, and Alexandru Telea. Computing refined skeletal features from medial point clouds. *Pattern Recognition Letters*, 76:13–21, 2016.

[114] Per Erik Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14(3):227–248, 1980.

[115] Yaorong Ge and J Michael Fitzpatrick. On the generation of skeletons from discrete euclidean distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1055–1066, 1996.

[116] Grégoire Malandain and Sara Fernández-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16(5):317–327, 1998.

[117] Chris Pudney. Distance-ordered homotopic thinning: a skeletonization algorithm for 3d digital images. *Computer Vision and Image Understanding*, 72(3):404–413, 1998.

[118] Ming Wan, Frank Dachille, and Arie Kaufman. Distance-field based skeletons for virtual navigation. In *Proceedings of the conference on Visualization'01*, pages 239–246. IEEE Computer Society, 2001.

[119] Kaleem Siddiqi, Sylvain Bouix, Allen Tannenbaum, and Steven W Zucker. Hamilton-jacobi skeletons. *International Journal of Computer Vision*, 48(3):215–231, 2002.

[120] Wim H Hesselink and Jos BTM Roerdink. Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(12):2204–2217, 2008.

[121] Ron Kimmel, Doron Shaked, Nahum Kiryati, and Alfred M Bruckstein. Skeletonization via distance maps and level sets. *Computer vision and image understanding*, 62(3):382–391, 1995.

[122] Martin Rumpf and Alexandru Telea. A continuous skeletonization method based on level sets. In *Proceedings of the symposium on Data Visualisation 2002*, pages 151–ff. Eurographics Association, 2002.

[123] Narendra Ahuja and Jen-Hui Chuang. Shape representation using a generalized potential field model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):169–176, 1997.

[124] Rui Ma and Tie-Ru Wu. Computing hierarchical curve-skeletons of 3d objects based on generalized potential field [j]. *Journal of Computer Applications*, 1:003, 2011.

[125] Nicu D Cornea, Deborah Silver, Xiaosong Yuan, and Raman Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer*, 21(11):945–955, 2005.

[126] Mohamed Sabry Hassouna and Aly A Farag. Variational curve skeletons using gradient vector flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2257–2274, 2009.

[127] Denis Rutovitz. Pattern recognition. *Journal of the Royal Statistical Society*, 129:504–530, 1966.

[128] YF Tsao and King Sun Fu. A parallel thinning algorithm for 3-d pictures. *Computer graphics and image processing*, 17(4):315–331, 1981.

[129] Jun-Sik Kwon, Jun-Woong Gi, and Eung-Kwan Kang. An enhanced thinning algorithm using parallel processing. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 752–755. IEEE, 2001.

[130] Chyi-Jou Gau and T Yung Kong. Minimal non-simple sets in 4d binary images. *Graphical Models*, 65(1):112–130, 2003.

[131] Paweł Dłotko and Ruben Specogna. Topology preserving thinning of cell complexes. *IEEE Transactions on Image Processing*, 23(10):4486–4495, 2014.

[132] Steven Lobregt, Piet W Verbeek, and Frans CA Groen. Three-dimensional skeletonization: principle and algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-2(1):75–77, 1980.

[133] David G Morgenthaler. Three-dimensional digital topology: The genus. Technical report, MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB, 1980.

[134] Hanno Homann. Implementation of a 3d thinning algorithm. *Insight Journal*, 421, 2007.

[135] GrabCAD community. Grabcad. https://grabcad.com.

[136] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[137] Ali Kaveh. *Structural mechanics: graph and matrix methods*, volume 6. Macmillan International Higher Education, 1992.

[138] Holger Keitel, Ghada Karaki, Tom Lahmer, Susanne Nikulla, and Volkmar Zabel. Evaluation of coupled partial models in structural engineering using graph theory and sensitivity analysis. *Engineering structures*, 33(12):3726–3736, 2011.

[139] Alexandre X Falcão, Jorge Stolfi, and Roberto de Alencar Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):19–29, 2004.

[140] Avneesh Sud, Mark Foskey, and Dinesh Manocha. Homotopy-preserving medial axis simplification. *International Journal of Computational Geometry & Applications*, 17(05):423–451, 2007.

[141] Dennie Reniers, Jarke Van Wijk, and Alexandru Telea. Computing multiscale curve and surface skeletons of genus 0 shapes using a global importance measure. *IEEE Transactions on Visualization and Computer Graphics*, 14(2), 2008.

[142] Balint Miklos, Joachim Giesen, and Mark Pauly. Discrete scale axis representations for 3d geometry. *ACM Transactions on Graphics (TOG)*, 29(4):101, 2010.

[143] D Ebert, P Brunet, and I Navazo. An augmented fast marching method for computing skeletons and centerlines. *Proceedings of VisSym, Barcelona, Spain*, 2002.

[144] Doron Shaked and Alfred M Bruckstein. Pruning medial axes. *Computer vision and image understanding*, 69(2):156–169, 1998.

[145] Mark Foskey, Ming C Lin, and Dinesh Manocha. Efficient computation of a simplified medial axis. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 96–107. ACM, 2003.

[146] Sylvain Bouix, Kaleem Siddiqi, and Allen Tannenbaum. Flux driven automatic centerline extraction. *Medical image analysis*, 9(3):209–221, 2005.

[147] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 85–94. Citeseer, 1998.

[148] Ian Stroud. *Boundary representation modelling techniques*. Springer Science & Business Media, 2006.

[149] G Zavarise and P Wriggers. Contact with friction between beams in 3-d space. *International Journal for Numerical Methods in Engineering*, 49(8):977–1006, 2000.

[150] Oliver Weeger, Bharath Narayanan, Laura De Lorenzis, Josef Kiendl, and Martin L Dunn. An isogeometric collocation method for frictionless contact of cosserat rods. *Computer Methods in Applied Mechanics and Engineering*, 321:361–382, 2017.

[151] Ashish Gupta, George Allen, and Jarek Rossignac. Quador: Quadric-of-revolution beams for lattices. *Computer-Aided Design*, 102:160–170, 2018.

[152] Ashish Gupta, George Allen, and Jarek Rossignac. Exact representations and geometric queries for lattice structures with quador beams. *Computer-Aided Design*, 115:64–77, 2019.

[153] Fehmi Cirak and Malcolm Sabin. Filletting quador lattice structures. *arXiv preprint arXiv:1908.06974*, 2019.

[154] K. Bandara, F. Cirak, G. Of, O. Steinbach, and J. Zapletal. Boundary element based multiresolution shape optimisation in electrostatics. *Journal of Computational Physics*, 297:584–598, 2015.

[155] C Bradford Barber, David P Dobkin, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[156] Zhongping Ji, Ligang Liu, and Yigang Wang. B-mesh: a modeling system for base meshes of 3d articulated shapes. In *Computer Graphics Forum*, volume 29, pages 2169–2177. Wiley Online Library, 2010.

[157] Athina Panotopoulou, Elissa Ross, Kathrin Welker, Evelyne Hubert, and Geraldine Morin. Scaffolding a skeleton. In *Research in Shape Analysis*, pages 17–35. Springer, 2018.

[158] Aristides G Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys (CSUR)*, 12(4):437–464, 1980.

[159] Christopher J Smith, Matthew Gilbert, Iain Todd, and Fatos Derguti. Application of layout optimization to the design of additively manufactured metallic components. *Structural and Multidisciplinary Optimization*, 54(5):1297–1313, 2016.

[160] Rahul Arora, Alec Jacobson, Timothy R. Langlois, Yijiang Huang, Caitlin Mueller, Wojciech Matusik, Ariel Shamir, Karan Singh, and David I.W. Levin. Volumetric michell trusses for parametric design & fabrication. In *Proceedings of the 3rd ACM Symposium on Computation Fabrication*, SCF '19, 2019.

[161] Robert McNeel & Associates. Rhinoceros. https://www.rhino3d.com.

[162] FreeCAD community. FreeCAD. https://www.freecadweb.org.

[163] Steven G. Johnson. The NLopt nonlinear-optimization package. http://github.com/stevengj/nlopt.

[164] Pierre Bézier. *Numerical control: mathematics and applications*. Wiley, 1970.

[165] Philip J Schneider. An algorithm for automatically fitting digitized curves. In *Graphics gems*, pages 612–626. Academic Press Professional, Inc., 1990.

[166] Julian AT Dow. *A unified approach to the finite element method and error analysis procedures*. Elsevier, 1998.

[167] Xiao Xiao, Malcolm Sabin, and Fehmi Cirak. Interrogation of spline surfaces with application to isogeometric design and analysis of lattice-skin structures. *Computer Methods in Applied Mechanics and Engineering*, 351:928–950, 2019.

[168] Altair. Altair Inspire. https://solidthinking.com/.

[169] Dassault Systèmes. Abaqus CAE. https://www.3ds.com/.

[170] Erin Bradner, Francesco Iorio, and Mark Davis. Parameters tell the design story: ideation and abstraction in design optimization. In *Proceedings of the symposium on simulation for architecture & urban design*, page 26. Society for Computer Simulation International, 2014.

[171] Jikai Liu and Yongsheng Ma. A survey of manufacturing oriented topology optimization methods. *Advances in Engineering Software*, 100:161–175, 2016.

[172] ANSYS. Ansys Discovery Live. https://www.ansys.com/.

[173] PTC. Creo simulation live. https://www.ptc.com/.

[174] Thomas Henry Gordon Megson. *Aircraft structures for engineering students*. Butterworth-Heinemann, 2016.

[175] Mehmet Akgun, Raphael Haftka, K Wu, and Joanne Walsh. Sensitivity of lumped constraints using the adjoint method. In *40th Structures, Structural Dynamics, and Materials Conference and Exhibit*, page 1314, 1999.

[176] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[177] Claude Fleury and Vincent Braibant. Structural optimization: a new dual method using mixed variables. *International journal for numerical methods in engineering*, 23(3):409–428, 1986.

APPENDIX A

Member stiffness matrix

A.1 Linear hexahedron element stiffness matrix

For a solid linear hexahedron element, the constitutive matrix $\mathbf{C}$ is computed using the solid Young's modulus $\overline{E}$ with the density $\rho = 1$. According to generalised Hooke's Law, the constitutive matrix $\mathbf{C}$ is given as

$$\mathbf{C} = \frac{\overline{E}}{(1+\nu)(1-2\nu)} \times \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{pmatrix}, \tag{A.1}$$

where ν is the Poisson ratio of the material used in the element. The element stiffness matrix $\overline{\mathbf{K}}_i$ of element i can be computed by integrating with respect to the element domain Ω as

$$\overline{\mathbf{K}}_i = \int_\Omega \mathbf{B}^\mathsf{T} \mathbf{C} \mathbf{B} \, \mathrm{d}\,\Omega \tag{A.2}$$

Where $\mathbf{B}$ is the strain-displacement matrix relating the strain $\boldsymbol{\epsilon}$ and the nodal displacement $\mathbf{u}$, $\boldsymbol{\epsilon} = \mathbf{B}\,\mathbf{u}$, which reads

$$\mathbf{B} = \begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \dots & \frac{\partial N_8}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \dots & 0 & \frac{\partial N_8}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & \dots & 0 & 0 & \frac{\partial N_8}{\partial z} \\ \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} & 0 & \dots & \frac{\partial N_8}{\partial x} & \frac{\partial N_8}{\partial y} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial z} & \dots & 0 & \frac{\partial N_8}{\partial y} & \frac{\partial N_8}{\partial z} \\ \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_1}{\partial z} & \dots & \frac{\partial N_8}{\partial x} & 0 & \frac{\partial N_8}{\partial z} \end{pmatrix}, \tag{A.3}$$

where N_i,$i = 1, \dots, 8$ are shape functions associated with the eight nodes of one hexahedral element, and x, y, z are the global coordinates. The result stiffness matrix is symmetric and with 64 entries.

Especially for a unit linear hexahedral element, which is commonly seen in most topology optimisation packages due to its simplicity, (A.2) is written as

$$\mathbf{K}_i = \frac{\overline{E}}{(1+\nu)(1-2\nu)} \begin{pmatrix} \mathbf{K}_1 & \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_4 \\ & \mathbf{K}_5 & \mathbf{K}_6 & \mathbf{K}_4 \\ sym. & & \mathbf{K}_5 & \mathbf{K}_2 \\ & & & \mathbf{K}_1 \end{pmatrix}, \tag{A.4}$$

where the six blocks in (A.4) are 6×6 symmetric matrices, which read

$$\mathbf{K}_1 = \begin{pmatrix} k_1 & k_2 & k_2 & k_3 & k_5 & k_5 \\ & k_1 & k_2 & k_4 & k_6 & k_7 \\ & & k_1 & k_4 & k_7 & k_6 \\ & & & k_1 & k_8 & k_8 \\ & sym. & & & k_1 & k_2 \\ & & & & & k_1 \end{pmatrix}, \quad \mathbf{K}_2 = \begin{pmatrix} k_9 & k_8 & k_{12} & k_6 & k_4 & k_7 \\ & k_9 & k_{12} & k_5 & k_3 & k_5 \\ & & k_{13} & k_7 & k_4 & k_6 \\ & & & k_9 & k_2 & k_{10} \\ & sym. & & & k_9 & k_{12} \\ & & & & & k_{13} \end{pmatrix},$$

$$\mathbf{K}_3 = \begin{pmatrix} k_6 & k_7 & k_4 & k_9 & k_{12} & k_8 \\ & k_6 & k_4 & k_{10} & k_{13} & k_{10} \\ & & k_3 & k_8 & k_{12} & k_9 \\ & & & k_6 & k_{11} & k_5 \\ & sym. & & & k_6 & k_4 \\ & & & & & k_3 \end{pmatrix}, \quad \mathbf{K}_4 = \begin{pmatrix} k_{14} & k_{11} & k_{11} & k_{13} & k_{10} & k_{10} \\ & k_{14} & k_{11} & k_{12} & k_9 & k_8 \\ & & k_{14} & k_{12} & k_8 & k_9 \\ & & & k_{14} & k_7 & k_7 \\ & sym. & & & k_{14} & k_{11} \\ & & & & & k_{14} \end{pmatrix},$$

$$\mathbf{K}_5 = \begin{pmatrix} k_1 & k_2 & k_8 & k_3 & k_5 & k_4 \\ & k_1 & k_8 & k_4 & k_6 & k_{11} \\ & & k_1 & k_5 & k_{11} & k_6 \\ & & & k_1 & k_8 & k_2 \\ & sym. & & & k_1 & k_8 \\ & & & & & k_1 \end{pmatrix}, \quad \mathbf{K}_6 = \begin{pmatrix} k_{14} & k_{11} & k_7 & k_{13} & k_{10} & k_{12} \\ & k_{14} & k_7 & k_{12} & k_9 & k_2 \\ & & k_{14} & k_{10} & k_2 & k_9 \\ & & & k_{14} & k_7 & k_{11} \\ & sym. & & & k_{14} & k_7 \\ & & & & & k_{14} \end{pmatrix},$$

and

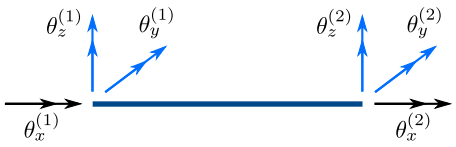
$$k_1 = -\frac{6\nu - 4}{9}, \quad k_2 = \frac{1}{12}, \quad k_3 = -\frac{1}{9}, \quad k_4 = -\frac{4\nu - 1}{12}, \quad k_5 = \frac{4\nu - 1}{12},$$
$$k_6 = \frac{1}{18}, \quad k_7 = \frac{1}{24}, \quad k_8 = -\frac{1}{12}, \quad k_9 = \frac{6\nu - 5}{36}, \quad k_{10} = -\frac{4\nu - 1}{24},$$
$$k_{11} = -\frac{1}{24}, \quad k_{12} = \frac{4\nu - 1}{24}, \quad k_{13} = -\frac{3\nu - 1}{18}, \quad k_{14} = \frac{3\nu - 2}{18}.$$

A.2 Timoshenko beam element stiffness matrix

The members of the structural frame are modelled as Timoshenko beams. As mentioned, the members are assumed to be straight and have uniform cross-sections along their lengths. Hence, each member can be approximated with a single beam finite element and its stiffness matrix can be derived with standard structural analysis techniques from undergraduate textbooks, see e.g. [174]. The stiffness matrix used here takes into account axial stretch, transversal shear, bending and torsion effects. The local stiffness matrix $\mathbf{K}_i^l$ of a beam i is derived in a local coordinate system in which the beam axis is aligned with the x_i^l axis. Each of the two end nodes of the beam have three displacement and three rotation degrees of freedom, see Figure A.1. To ease the notation, here writes in the following for the local coordinate axes only x, y and z.

The element stiffness matrix of the beam is composed out of independent axial, bending and torsion stiffness matrices. The axial stiffness matrix corresponding to the

(a) Displacement degrees of freedom



(b) Rotation degrees of freedom

Fig. A.1 Degrees of freedom of a Timoshenko beam element. Coupled degrees of freedom are depicted in the same colours.

displacement degrees of freedom $u_x^{(1)}$ and $u_x^{(2)}$ is given by

$$\frac{EA}{L}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{A.5}$$

where E, A and L denote the Young's modulus, cross-sectional area and length.

The bending stiffness matrix corresponding to the displacement degrees of freedom $u_y^{(1)}$, $u_z^{(1)}$, $u_y^{(2)}$ and $u_z^{(2)}$ and the rotational degrees of freedom $\theta_y^{(1)}$, $\theta_z^{(1)}$, $\theta_y^{(2)}$ and $\theta_z^{(2)}$ is given by

$$\begin{pmatrix}
\frac{12EI_z}{(1+b_z)L^3} & 0 & 0 & \frac{6EI_z}{(1+b_z)L^2} & -\frac{12EI_z}{(1+b_z)L^3} & 0 & 0 & \frac{6EI_z}{(1+b_z)L^2} \\
 & \frac{12EI_y}{(1+b_y)L^3} & -\frac{6EI_y}{(1+b_y)L^2} & 0 & 0 & -\frac{12EI_y}{(1+b_y)L^3} & -\frac{6EI_y}{(1+b_y)L^2} & 0 \\
 & & \frac{(4+b_y)EI_y}{(1+b_y)L} & 0 & 0 & \frac{6EI_y}{(1+b_y)L^2} & \frac{(2-b_y)EI_y}{(1+b_y)L} & 0 \\
 & & & \frac{(4+b_z)EI_z}{(1+b_z)L} & -\frac{6EI_z}{(1+b_z)L^2} & 0 & 0 & \frac{(2-b_z)EI_z}{(1+b_z)L} \\
 & & & & \frac{12EI_z}{(1+b_z)L^3} & 0 & 0 & -\frac{6EI_z}{(1+b_z)L^2} \\
 & & sym. & & & \frac{12EI_y}{(1+b_y)L^3} & \frac{6EI_y}{(1+b_y)L^2} & 0 \\
 & & & & & & \frac{(4+b_y)EI_y}{(1+b_y)L} & 0 \\
 & & & & & & & \frac{(4+b_z)EI_z}{(1+b_z)L}
\end{pmatrix}, \tag{A.6}$$

where I_y and I_z are the second moments of area about the y and z axis. The two dimensionless factors b_y and b_z take into account the transversal shear and are defined according to $b = 12EI/(\kappa GAL^2)$, with G the shear modulus and κ the shear correction factor. In case of a circular cross section, $I_y = I_z = I$, $b_y = b_z = b$ and $\kappa = 0.9$. Note that setting $b_y = b_z = 0$ in (A.6) gives the stiffness matrix of an Euler-Bernoulli beam.

Finally, the torsional stiffness matrix corresponding to the rotational degrees of freedom $\theta_x^{(1)}$ and $\theta_x^{(2)}$ is given by

$$\frac{GJ}{L}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{A.7}$$

where J is the torsion constant of the cross section.

The local stiffness matrix $\mathbf{K}_i^l$ of an element i is obtained by suitably assembling the three stiffness matrices (A.5), (A.6) and (A.7) into a 12×12 matrix. This matrix in the local coordinate system x_i^l, y_i^l and z_i^l is transformed into a stiffness matrix $\mathbf{K}_i$ in the global coordinate system x, y and z according to (2.24).

The local coordinate system is transformed into global system using two elemental rotation angles, α_i around the z_i^l axis and β_i around y_i^l axis. These two spatial angles are illustrated in Figure A.2. Thus, two spacial angles α_i and β_i in (2.26) are determined as

$$\alpha_i = \text{atan}\left(\frac{y_i^{(2)} - y_i^{(1)}}{x_i^{(2)} - x_i^{(1)}}\right), \text{ and} \tag{A.8a}$$

$$\beta_i = \text{atan}\left(\frac{z_i^{(2)} - z_i^{(1)}}{\sqrt{(x_i^{(2)} - x_i^{(1)})^2 + (y_i^{(2)} - y_i^{(1)})^2}}\right), \tag{A.8b}$$

where x_I, $y_i^{(1)}$, $z_i^{(1)}$ and $x_i^{(2)}$, $y_i^{(2)}$, $z_i^{(2)}$ are global coordinates of the two nodes of the beam element i, and the beam has length of

$$L_i = \sqrt{(x_i^{(2)} - x_i^{(1)})^2 + (y_i^{(2)} - y_i^{(1)})^2 + (z_i^{(2)} - z_i^{(1)})^2}\,. \tag{A.9}$$

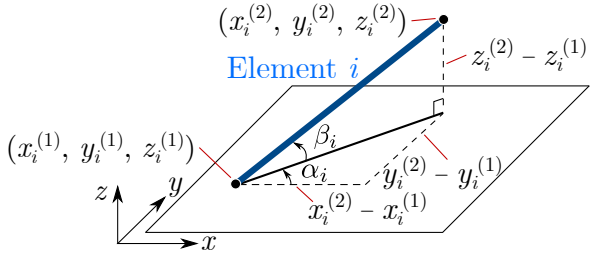


Fig. A.2 Spatial angles α_i and β_i. The beam element i coloured in dark blue lies in a global coordinate system x, y, z; it starts from node with coordinate of $(x_i^{(1)}, y_i^{(1)}, z_i^{(1)})$ and ends at $(x_i^{(2)}, y_i^{(2)}, z_i^{(2)})$. A $x - y$ plane is drawn passing through the starting node of the beam. The black solid line on the plane is the projectile of beam onto the x-y plane.

APPENDIX B

KREISSELMEIER-STEINHAUSER AGGREGATION FUNCTION

Presented by Kreisselmeier and Steinhauser [84], KS function is a widely-used constraint aggregation method for gradient-based optimization [85, 175]. KS function can wrap the constraint into a explicit function with explicit derivatives in order to deal with the constraints that cannot be explicitly or simply expressed. In this thesis, the cost function of frame optimisation is always only the structural compliance, thus here the KS function aggregation in optimisation with only one cost function is discussed. But to notice, KS function can be used for multiple cost function optimisations, for example, the optimisation under multiple loading cases. Without losing generality, assume there is a optimisation problem

$$\begin{aligned} \text{minimise} \quad & F(\boldsymbol{x}) && \text{(B.1a)} \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leq 0\,, \quad i \in \{1, \ldots, n\} && \text{(B.1b)} \\ & \boldsymbol{x} \in \mathbb{R}^m, && \text{(B.1c)} \end{aligned}$$

Where $F(\boldsymbol{x})$ is the cost function associated with constraints of $g_i(\boldsymbol{x}) \leq 0$, $i = 1, \ldots, n$ and optimisation variable $\boldsymbol{x} = \{x_1, \ldots, x_m\}$. Constraints $g_i(\boldsymbol{x})$ are wrapped into KS function as

$$KS(g_i(\boldsymbol{x})) = \frac{1}{\tau} \ln \sum_{i}^{n} e^{\tau g_i(\boldsymbol{x})}, \tag{B.2}$$

where τ is the aggregation variable (typically $5 < \tau < 200$). The first derivative of KS function (B.2) with respect to optimisation variable can be derived as

$$\frac{\partial KS(g_i(\boldsymbol{x}))}{\partial x_j} = \frac{\sum_i^n \left(e^{\tau g_i(\boldsymbol{x})} \frac{\partial g_i(\boldsymbol{x})}{\partial x_j} \right)}{\sum_i^n e^{\tau g_i(\boldsymbol{x})}} \tag{B.3}$$

Alternatively there is a modified version of Equation (B.2) for better numerical stability if the maximum constraint function value, $g_{\max}$, is known, which reads

$$KS(g_i(\boldsymbol{x})) = g_{\max} + \frac{1}{\tau} \ln \sum_n^i e^{\tau (g_i(\boldsymbol{x}) - g_{\max})}. \tag{B.4}$$

The first derivative of Equation (B.4) with respect to optimisation variable is

$$\frac{\partial KS(g_i(\boldsymbol{x}))}{\partial x_j} = \frac{\sum_i^n \left(e^{\tau (g_i(\boldsymbol{x}) - g_{\max})} \frac{\partial g_i(\boldsymbol{x})}{\partial x_j} \right)}{\sum_i^n e^{\tau (g_i(\boldsymbol{x}) - g_{\max})}}. \tag{B.5}$$

Then, the optimisation problem (B.1) becomes

$$\begin{aligned}
&\text{minimise} && F(\boldsymbol{x}) && \text{(B.6a)}\\
&\text{subject to} && KS(g_i(\boldsymbol{x})) \leq 0\,, \quad i \in \{1, \ldots, n\} && \text{(B.6b)}\\
& && \boldsymbol{x} \in \mathbb{R}^m. && \text{(B.6c)}
\end{aligned}$$

The previous optimisation problem (B.1) has n constraint functions, whereas in simplified optimisation problem (B.6), all n constraint functions are wrapped into one KS function.

APPENDIX C

Gradient-based optimisation algorithms

C.1 Sequential quadratic programming

The basic idea of sequential quadratic programming (SQP) is to approximate the nonlinear optimisation problem at each iteration with a quadratic programming subproblem and get a better approximation based on the solution to this subproblem. This process leads to a sequence of quadratic programming subproblems, and the iteration continues until some termination condition is satisfied.

Consider that $\{\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\}$ is the solution at the k-th iteration, and it is assumed that the solution in the next iteration $\{\boldsymbol{x}^{(k+1)}, \boldsymbol{\lambda}^{(k+1)}, \boldsymbol{\phi}^{(k+1)}\}$ is closer to the local minimum $\{\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\phi}^*\}$. The quadratic Taylor series approximation in $\boldsymbol{x}$ for the Lagrangian at the $(k+1)$-th iteration is

$$\begin{aligned}\mathcal{L}\Big(\boldsymbol{x}^{(k+1)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big) \approx \mathcal{L}\Big(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big) + \boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} \mathcal{L}\Big(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big) \\ + \frac{1}{2} \boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}}^2 \mathcal{L}\Big(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big) \boldsymbol{\delta}_{\boldsymbol{x}} \,.\end{aligned} \tag{C.1}$$

It is reasonable to consider (C.1) as the objective function in the quadratic subproblem for the next iteration. Since $\mathcal{L}\Big(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big)$ is a constant at the $(k+1)$-th iteration, the second and the third terms in the Taylor approximation (C.1) can then be considered as the objective function, i.e.

$$\boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} \mathcal{L}\Big(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big) + \frac{1}{2} \boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}}^2 \mathcal{L}\Big(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\Big) \boldsymbol{\delta}_{\boldsymbol{x}} \,. \tag{C.2}$$

In practice, the Hessian of the Lagrangian $\nabla^2_{\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right)$ is approximated by the Hessian of the original objective function $\nabla^2_{\boldsymbol{x}} J\left(\boldsymbol{x}^{(k)}\right)$ in order to be able to solve the quadratic subproblem at any $\boldsymbol{x}^{(k)}$ and easier for a global convergence analysis [75]. However, the Hessian is not easy to compute in general as it involves the second derivatives of the objective and constraint functions. To cope with this issue, the exact Hessian is usually replaced with a quasi-Newton approximation [176]. In addition, $\nabla_{\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right)$ is replaced by $\nabla_{\boldsymbol{x}} J\left(\boldsymbol{x}^{(k)}\right)$. As a matter of fact, this replacement leads to an equivalent subproblem when only equality constraints and their linearised approximations are considered. If inequality constraints exist, the replacement is not quite equivalent though it leads to an equivalent subproblems when the slack-variable formulation of the original optimisation problem is considered [75].

The linearised approximations of the constraints are considered with the first term of the Taylor series in $\boldsymbol{x}$

$$G_i\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) \approx G_i\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) + \delta_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} G_i\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) , \tag{C.3}$$

$$H_j\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) \approx H_j\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) + \delta_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} H_j\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) . \tag{C.4}$$

Therefore, the quadratic subproblem at the $(k+1)$-th iteration is of the form

$$\text{minimise} \quad \boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} J\left(\boldsymbol{x}^{(k)}\right) + \frac{1}{2} \boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla^2_{\boldsymbol{x}} J\left(\boldsymbol{x}^{(k)}\right) \boldsymbol{\delta}_{\boldsymbol{x}} \tag{C.5a}$$

$$\text{subject to} \quad G_i\left(\boldsymbol{x}^{(k)}\right) + \delta_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} G_i\left(\boldsymbol{x}^{(k)}\right) = 0, \quad i = 1, \cdots, n_p \tag{C.5b}$$

$$H_j\left(\boldsymbol{x}^{(k)}\right) + \delta_{\boldsymbol{x}}^{\mathsf{T}} \nabla_{\boldsymbol{x}} H_j\left(\boldsymbol{x}^{(k)}\right) \leq 0, \quad j = 1, \cdots, n_q \tag{C.5c}$$

The solution $\boldsymbol{\delta}^*_{\boldsymbol{x}}$ of the quadratic subproblem (C.5) can be used to calculate the next iterate $\boldsymbol{x}^{(k+1)}$ by updating $\boldsymbol{x}^{(k)}$ in the direction of $\boldsymbol{\delta}^*_{\boldsymbol{x}}$. For the new iterates of multipliers, one possible set of candidates are the corresponding optimal multipliers of (C.5). The validity of using the optimal multipliers of the quadratic subproblem can be seen from the following analysis.

Considering the linear approximation of the KKT condition [80] yields,

$$\begin{aligned}\nabla_{\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{\lambda}^{(k+1)}, \boldsymbol{\phi}^{(k+1)}\right) \approx & \nabla_{\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) + \boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}} \nabla^2_{\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) \\ & + \boldsymbol{\delta}_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla^2_{\boldsymbol{\lambda}\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right) + \boldsymbol{\delta}_{\boldsymbol{\phi}}^{\mathsf{T}} \nabla^2_{\boldsymbol{\phi}\boldsymbol{x}}\mathcal{L}\left(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\phi}^{(k)}\right)\end{aligned} \tag{C.6}$$

with

$$\nabla_{\boldsymbol{x}}\mathcal{L}\Big(\boldsymbol{x}^{(k)},\boldsymbol{\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\Big)=\nabla_{\boldsymbol{x}}J\Big(\boldsymbol{x}^{(k)}\Big)+\sum_{i=1}^{n_p}\lambda_i^{(k)}\nabla_{\boldsymbol{x}}G_i\Big(\boldsymbol{x}^{(k)}\Big)+\sum_{j=1}^{n_q}\phi_j^{(k)}\nabla_{\boldsymbol{x}}H_j\Big(\boldsymbol{x}^{(k)}\Big)\,, \tag{C.7}$$

$$\nabla^2_{\boldsymbol{x}}\mathcal{L}\Big(\boldsymbol{x}^{(k)},\boldsymbol{\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\Big)=\nabla^2_{\boldsymbol{x}}J\Big(\boldsymbol{x}^{(k)}\Big)+\sum_{i=1}^{n_p}\lambda_i^{(k)}\nabla^2_{\boldsymbol{x}}G_i\Big(\boldsymbol{x}^{(k)}\Big)+\sum_{j=1}^{n_q}\phi_j^{(k)}\nabla^2_{\boldsymbol{x}}H_j\Big(\boldsymbol{x}^{(k)}\Big)\,, \tag{C.8}$$

$$\nabla^2_{\boldsymbol{\lambda}\boldsymbol{x}}\mathcal{L}\Big(\boldsymbol{x}^{(k)},\boldsymbol{\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\Big)=\nabla_{\boldsymbol{x}}\boldsymbol{G}\Big(\boldsymbol{x}^{(k)}\Big)=\Big(\nabla_{\boldsymbol{x}}G_1(\boldsymbol{x}^{(k)})\,\nabla_{\boldsymbol{x}}G_2(\boldsymbol{x}^{(k)})\,\cdots\,\nabla_{\boldsymbol{x}}G_{n_p}(\boldsymbol{x}^{(k)})\Big)^{\mathsf{T}}\,, \tag{C.9}$$

$$\nabla^2_{\boldsymbol{\phi}\boldsymbol{x}}\mathcal{L}\Big(\boldsymbol{x}^{(k)},\boldsymbol{\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\Big)=\nabla_{\boldsymbol{x}}\boldsymbol{H}\Big(\boldsymbol{x}^{(k)}\Big)=\Big(\nabla_{\boldsymbol{x}}H_1(\boldsymbol{x}^{(k)})\,\nabla_{\boldsymbol{x}}H_2(\boldsymbol{x}^{(k)})\,\cdots\,\nabla_{\boldsymbol{x}}H_{n_q}(\boldsymbol{x}^{(k)})\Big)^{\mathsf{T}}\,. \tag{C.10}$$

Substituting (C.7), (C.8), (C.9) and (C.10) into (C.6) yields

$$\nabla_{\boldsymbol{x}}J\Big(\boldsymbol{x}^{(k)}\Big)+\boldsymbol{\delta}_{\boldsymbol{x}}^{\mathsf{T}}\nabla^2_{\boldsymbol{x}}\mathcal{L}\Big(\boldsymbol{x}^{(k)},\boldsymbol{\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\Big)+\boldsymbol{\lambda}_{k+1}^{\mathsf{T}}\nabla_{\boldsymbol{x}}\boldsymbol{G}\Big(\boldsymbol{x}^{(k)}\Big)+\boldsymbol{\phi}_{k+1}^{\mathsf{T}}\nabla_{\boldsymbol{x}}\boldsymbol{H}\Big(\boldsymbol{x}^{(k)}\Big)=0\,. \tag{C.11}$$

Given $\Big(\boldsymbol{x}^{(k)},\boldsymbol{\lambda}^{(k)},\boldsymbol{\phi}^{(k)}\Big)$, (C.11) is exactly the first derivative condition of the quadratic subproblem (C.5) at a local minimum $\boldsymbol{\delta}_{\boldsymbol{x}}^*$ when the Hessian of the Lagrangian is approximated with the Hessian of the original objective function. Therefore, the optimal multipliers $\{\boldsymbol{\lambda}_{k+1}^*,\boldsymbol{\phi}_{k+1}^*\}$ obtained from (C.5) is an appropriate choice for the next iterates of multipliers $\boldsymbol{\lambda}^{(k+1)}$ and $\boldsymbol{\phi}^{(k+1)}$. A modification of the next iterates $\{\boldsymbol{x}^{(k+1)},\boldsymbol{\lambda}^{(k+1)},\boldsymbol{\phi}^{(k+1)}\}$ would be a line search along the directions $\{\boldsymbol{\delta}_{\boldsymbol{x}},\boldsymbol{\delta}_{\boldsymbol{\lambda}},\boldsymbol{\delta}_{\boldsymbol{\phi}}\}$ where

$$\boldsymbol{\delta}_{\boldsymbol{x}}=\boldsymbol{\delta}_{\boldsymbol{x}}^*,\quad \boldsymbol{\delta}_{\boldsymbol{\lambda}}=\boldsymbol{\lambda}_{k+1}^*-\boldsymbol{\lambda}^{(k)},\quad \boldsymbol{\delta}_{\boldsymbol{\phi}}=\boldsymbol{\phi}_{k+1}^*-\boldsymbol{\phi}^{(k)}\,. \tag{C.12}$$

Therefore, the next iterates at $(k+1)$-th iteration are

$$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+\varepsilon^{(k)}\boldsymbol{\delta}_{\boldsymbol{x}},\quad \boldsymbol{\lambda}^{(k+1)}=\boldsymbol{\lambda}^{(k)}+\varepsilon^{(k)}\boldsymbol{\delta}_{\boldsymbol{\lambda}},\quad \boldsymbol{\phi}^{(k+1)}=\boldsymbol{\phi}^{(k)}+\varepsilon^{(k)}\boldsymbol{\delta}_{\boldsymbol{\phi}}\,, \tag{C.13}$$

where $\varepsilon^{(k)}$ is the step length which can be obtained by a line search.

One advantage of using SQP to solve nonlinear optimisation problems is that neither the initial point $\boldsymbol{x}_0$ nor the iterates $\boldsymbol{x}^{(k)}$ need to be feasible. In addition, SQP decomposes the nonlinear problem into a sequence of quadratic programming subproblems with linear constraints, which are relatively easier to solve as there are some good algorithms for the quadratic programming problems [176].

C.2 Method of moving asymptotes

The method of moving asymptotes (MMA) was first proposed by Svanberg [76]. It can be seen as a generalisation of the convex linearisation method developed by Fleury et al. [177]. In the MMA, each function at every iteration is linearised in terms of the intervening variables $1/\left(x_j - x_j^L\right)$ or $1/\left(x_j^U - x_j\right)$, where x_j^L and x_j^U are the moving asymptotes that can be used to control the conservation and stability of the optimisation process. The linearisation of functions in the MMA is as follows.

Given an iterate $\boldsymbol{x}^{(k)}$ at the k-th iteration, the values of moving asymptotes $x_j^{L(k)}$ and $x_j^{U(k)}$ are chosen such that

$$x_j^{L(k)} < x_j^{(k)} < x_j^{U(k)}, \quad j = 1, \cdots, n\,. \tag{C.14}$$

The linear approximation $f^{(k)}$ of a function f at the k-th iteration can be defined as

$$\begin{aligned} f^{(k)}(\boldsymbol{x}) &= f\left(\boldsymbol{x}^{(k)}\right) - \sum_{j=1}^{n}\left(\frac{f_j^{U(k)}}{x_j^{U(k)} - x_j^{(k)}} + \frac{f_j^{L(k)}}{x_j^{(k)} - x_j^{L(k)}}\right) + \sum_{j=1}^{n}\left(\frac{f_j^{U(k)}}{x_j^{U(k)} - x_j} + \frac{f_j^{L(k)}}{x_j - f_j^{L(k)}}\right) \\ &= f\left(\boldsymbol{x}^{(k)}\right) + \sum_{j=1}^{n} f_j^{U(k)}\left(\frac{1}{x_j^{U(k)} - x_j} - \frac{1}{x_j^{U(k)} - x_j^{(k)}}\right) \\ &\quad + \sum_{j=1}^{n} f_j^{L(k)}\left(\frac{1}{x_j - x_j^{L(k)}} - \frac{1}{x_j^{(k)} - x_j^{L(k)}}\right), \end{aligned} \tag{C.15}$$

where f can be either the objective function or constraint functions in the optimisation problem, and

$$f_j^{U(k)} = \begin{cases} \left(x_j^{U(k)} - x_j^{(k)}\right)^2 \dfrac{\partial f}{\partial x_j}, & \text{if } \dfrac{\partial f}{\partial x_j} > 0\,; \\ 0, & \text{otherwise}\,; \end{cases} \tag{C.16}$$

$$f_j^{L(k)} = \begin{cases} -\left(x_j^{(k)} - x_j^{L(k)}\right)^2 \dfrac{\partial f}{\partial x_j}, & \text{if } \dfrac{\partial f}{\partial x_j} < 0\,; \\ 0, & \text{otherwise}\,. \end{cases} \tag{C.17}$$

After substituting (C.16) and (C.17) into (C.15), the linear approximation in MMA becomes

$$f^{(k)}(\boldsymbol{x}) = \begin{cases} f\left(\boldsymbol{x}^{(k)}\right) + \sum_{j=1}^{n} \dfrac{f_j^{U(k)} - x_j^{(k)}}{f_j^{U(k)} - x_j} \dfrac{\partial f}{\partial x_j} \left(x_j - x_j^{(k)}\right), & \text{if } \dfrac{\partial f}{\partial x_j} > 0 \\ f\left(\boldsymbol{x}^{(k)}\right) + \sum_{j=1}^{n} \dfrac{x_j^{(k)} - f_j^{L(k)}}{x_j - f_j^{L(k)}} \dfrac{\partial f}{\partial x_j} \left(x_j - x_j^{(k)}\right), & \text{if } \dfrac{\partial f}{\partial x_j} < 0 \\ f\left(\boldsymbol{x}^{(k)}\right), & \text{if } \dfrac{\partial f}{\partial x_j} = 0 \end{cases} \tag{C.18}$$

It can be observed from (C.18) that $f^{(k)}$ is a linear approximation of f at $\boldsymbol{x}^{(k)}$. Specifically, $f^{(k)}\left(\boldsymbol{x}^{(k)}\right) = f\left(\boldsymbol{x}^{(k)}\right)$ and $\partial f^{(k)}/\partial x_j = \partial f/\partial x_j$ at $\boldsymbol{x} = \boldsymbol{x}^{(k)}$. The second derivatives of the approximation function $f^{(k)}$ are

$$\frac{\partial^2 f^{(k)}}{\partial x_j^2} = \frac{2 f_j^{U(k)}}{\left(x_j^{U(k)} - x_j\right)^3} + \frac{2 f_j^{L(k)}}{\left(x_j - x_j^{L(k)}\right)^3}, \tag{C.19}$$

$$\frac{\partial^2 f^{(k)}}{\partial x_j \partial x_k} = 0, \quad \text{if } j \neq k. \tag{C.20}$$

As can be seen, the second derivatives of $f^{(k)}$ are always nonnegative, which indicates that it is a convex function. Furthermore, the second derivatives at $\boldsymbol{x}^{(k)}$ become larger when the moving asymptotes $x_j^{L(k)}$ and $x_j^{U(k)}$ are closer to $\boldsymbol{x}^{(k)}$, indicating a larger curvature of the approximating function $f^{(k)}$ in the neighbourhood of $\boldsymbol{x}^{(k)}$. On the other hand, if $x_j^{L(k)}$ and $x_j^{U(k)}$ move far away from $\boldsymbol{x}^{(k)}$, $f^{(k)}$ becomes more linear around $\boldsymbol{x}^{(k)}$. For example in the extreme case where $x_j^{L(k)} = -\infty$ and $x_j^{U(k)} = \infty$, the approximating function $f^{(k)}$ becomes a linear function that is identical to the one adopted in the sequential linear programming.

The effect of the moving asymptotes $x_j^{L(k)}$ and $x_j^{U(k)}$ on the optimisation process is examined as follows. Consider two sets of moving asymptotes $\{x_j^{L(k)}, x_j^{U(k)}\}$ and $\{\widetilde{x}_j^{L(k)}, \widetilde{x}_j^{U(k)}\}$ satisfying

$$x_j^{L(k)} \leq \widetilde{x}_j^{L(k)} < x_j^{(k)} < \widetilde{x}_j^{U(k)} \leq x_j^{U(k)}, \tag{C.21}$$

the difference of the approximating functions $f^{(k)}$ and $\widetilde{f}^{(k)}$ defined in terms of $\{x_j^{L(k)}, x_j^{U(k)}\}$ and $\{\widetilde{x}_j^{L(k)}, \widetilde{x}_j^{U(k)}\}$ respectively is

$$\Delta f^{(k)} = \widetilde{f}^{(k)} - f^{(k)}. \tag{C.22}$$

It can be shown that $\Delta f^{(k)}(x_j^{(k)}) = 0$ and $\partial \Delta f^{(k)}/\partial x_j = 0$ at $x_j = x_j^{(k)}$. Recall that $\partial f^{(k)}/\partial x_j = \partial f/\partial x_j$ at $\boldsymbol{x} = \boldsymbol{x}^{(k)}$. Hence, $\partial f^{(k)}/\partial x_j = \partial \widetilde{f}^{(k)}/\partial x_j$ at $\boldsymbol{x} = \boldsymbol{x}^{(k)}$. The second derivative of $\Delta f^{(k)}$ with respect to x_j is

$$\left.\frac{\partial^2 \Delta f^{(k)}}{\partial x_j^2}\right|_{x_j = x_j^{(k)}} = \begin{cases} \dfrac{2}{\widetilde{x}_j^{U(k)} - x_j^{(k)}} \dfrac{\partial f}{\partial x_j} - \dfrac{2}{x_j^{U(k)} - x_j^{(k)}} \dfrac{\partial f}{\partial x_j}, & \text{if } \dfrac{\partial f}{\partial x_j} \geq 0 \\ -\dfrac{2}{x_j^{(k)} - \widetilde{x}_j^{L(k)}} \dfrac{\partial f}{\partial x_j} + \dfrac{2}{x_j^{(k)} - x_j^{L(k)}} \dfrac{\partial f}{\partial x_j}, & \text{if } \dfrac{\partial f}{\partial x_j} < 0 \end{cases} . \tag{C.23}$$

Since $\partial^2 \Delta f^{(k)}/\partial x_j^2$ is always nonnegative at $x_j = x_j^{(k)}$, $\boldsymbol{x}^{(k)}$ is a local minimum of $\Delta f^{(k)}$. Hence, $\Delta f^{(k)} \geq 0$, i.e. the function value approximated with $f^{(k)}$ is larger than the one approximated with $\widetilde{f}^{(k)}$. Therefore, using moving asymptotes closer to the iterate $\boldsymbol{x}^{(k)}$ leads to a more conservative approximation of the original function. This can be used to control the conservation and stability during the optimisation process by adjusting the values of moving asymptotes. If the optimisation process appears some oscillation, some closer moving asymptotes can be chosen; in the case of a slow convergence of the optimisation process, the asymptotes can be moved away from the current iterate.

The effect of moving asymptotes is illustrated in Figure C.1 plotting the function $f = x^2$ and its approximations f^M at $x = 1$ in the MMA considering different moving asymptote $x^U = 1.2, 1.5, 2$ and 10. It is shown that with a larger value of x^U, the approximation is more linear; on the other hand, the curvature of the approximation function is larger with a smaller value of x^U.

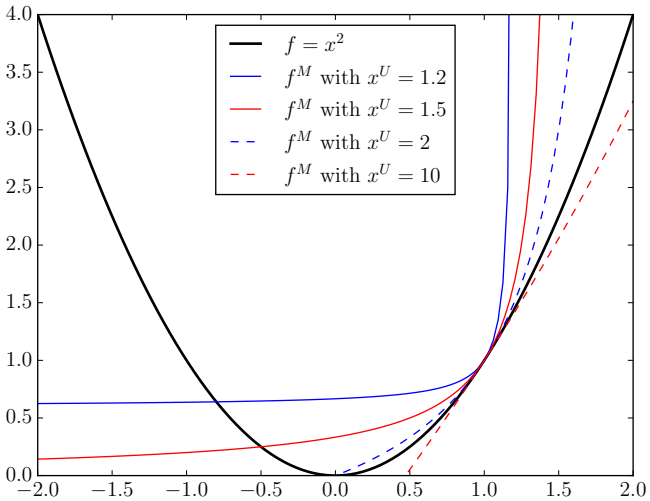


Fig. C.1 Approximating functions in MMA with different upper moving asymptote values

APPENDIX D

Euler characteristic look-up tables

The Euler characteristics look up table below are based on 26-neighbourhood. First, Table D.1 records the Euler characteristics of octants $\chi(Oct(vert))$ with the voxels numbering given on the left of the table, where *vert* is the centre vertex of the octant as shown as a black point below. The binary codes used in the table represent the configuration, and 1s indicate solid voxels while 0s mark where the empty voxels are.

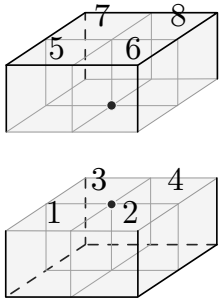


Table D.1 Look up table for Euler characteristics of octants

Binary code 8765 4321	$\chi(Oct)$	Binary code 8765 4321	$\chi(Oct)$
0000 0000	0	0001 1011	−1/4
0000 0001	1/8	0010 1011	−1/4
0000 0011	0	0000 1111	0
0000 1001	−1/4	0110 1011	3/8
1000 0001	−3/4	1011 0101	1/8
0000 0111	−1/8	0001 1111	−1/8
1010 0001	−3/8	1011 1101	1/4
0110 0001	−1/8	0101 1111	1/4
0110 1001	1/2	0011 1111	0
1100 0011	0	0111 1111	1/8
1110 0001	0	1111 1111	0

Second, Table D.2 stores the change in Euler characteristics when the voxel at position 1 is to be deleted. The change is denoted as $\bigtriangledown\chi = \bigtriangledown\chi(Oct(vert)) = \chi(Oct(vert) \setminus v) - \chi(Oct(vert))$, where v is the voxel to delete and vertex $vert \in v$.

Table D.2 Look up table for changes in Euler characteristics of octants

Binary code	$\bigtriangledown\chi$	Binary code	$\bigtriangledown\chi$	Binary code	$\bigtriangledown\chi$	Binary code	$\bigtriangledown\chi$	Binary code	$\bigtriangledown\chi$
0000 0001	$-1/8$	0011 0101	$-1/8$	0110 1001	$-5/8$	1001 1101	$-1/8$	1101 0001	$1/8$
0000 0011	$1/8$	0011 0111	$1/8$	0110 1011	$-3/8$	1001 1111	$1/8$	1101 0011	$-1/8$
0000 0101	$1/8$	0011 1001	$-3/8$	0110 1101	$-3/8$	1010 0001	$3/8$	1101 0101	$-1/8$
0000 0111	$-1/8$	0011 1011	$-1/8$	0110 1111	$-1/8$	1010 0011	$1/8$	1101 0111	$1/8$
0000 1001	$3/8$	0011 1101	$-1/8$	0111 0001	$1/8$	1010 0101	$-3/8$	1101 1001	$-3/8$
0000 1011	$1/8$	0011 1111	$1/8$	0111 0011	$-1/8$	1010 0111	$-1/8$	1101 1011	$-1/8$
0000 1101	$1/8$	0100 0001	$3/8$	0111 0101	$-1/8$	1010 1001	$-1/8$	1101 1101	$-1/8$
0000 1111	$-1/8$	0100 0011	$-3/8$	0111 0111	$1/8$	1010 1011	$1/8$	1101 1111	$1/8$
0001 0001	$-1/8$	0100 0101	$1/8$	0111 1001	$-3/8$	1010 1101	$-3/8$	1110 0001	$-1/8$
0001 0011	$-1/8$	0100 0111	$-1/8$	0111 1011	$-1/8$	1010 1111	$-1/8$	1110 0011	$-3/8$
0001 0101	$-1/8$	0100 1001	$-1/8$	0111 1101	$-1/8$	1011 0001	$1/8$	1110 0101	$-3/8$
0001 0111	$1/8$	0100 1011	$-3/8$	0111 1111	$1/8$	1011 0011	$-1/8$	1110 0111	$-1/8$
0001 1001	$-3/8$	0100 1101	$1/8$	1000 0001	$7/8$	1011 0101	$-1/8$	1110 1001	$-5/8$
0001 1011	$-1/8$	0100 1111	$-1/8$	1000 0011	$1/8$	1011 0111	$1/8$	1110 1011	$-3/8$
0001 1101	$-1/8$	0101 0001	$1/8$	1000 0101	$1/8$	1011 1001	$-3/8$	1110 1101	$-3/8$
0001 1111	$1/8$	0101 0011	$-1/8$	1000 0111	$-1/8$	1011 1011	$-1/8$	1110 1111	$-1/8$
0010 0001	$3/8$	0101 0101	$-1/8$	1000 1001	$3/8$	1011 1101	$-1/8$	1111 0001	$1/8$
0010 0011	$1/8$	0101 0111	$1/8$	1000 1011	$1/8$	1011 1111	$1/8$	1111 0011	$-1/8$
0010 0101	$-3/8$	0101 1001	$-3/8$	1000 1101	$1/8$	1100 0001	$3/8$	1111 0101	$-1/8$
0010 0111	$-1/8$	0101 1011	$-1/8$	1000 1111	$-1/8$	1100 0011	$-3/8$	1111 0111	$1/8$
0010 1001	$-1/8$	0101 1101	$-1/8$	1001 0001	$1/8$	1100 0101	$1/8$	1111 1001	$-3/8$
0010 1011	$1/8$	0101 1111	$1/8$	1001 0011	$-1/8$	1100 0111	$-1/8$	1111 1011	$-1/8$
0010 1101	$-3/8$	0110 0001	$-1/8$	1001 0101	$-1/8$	1100 1001	$-1/8$	1111 1101	$-1/8$
0010 1111	$-1/8$	0110 0011	$-3/8$	1001 0111	$1/8$	1100 1011	$-3/8$	1111 1111	$1/8$
0011 0001	$1/8$	0110 0101	$-3/8$	1001 1001	$-3/8$	1100 1101	$1/8$		
0011 0011	$-1/8$	0110 0111	$-1/8$	1001 1011	$-1/8$	1100 1111	$-1/8$		

APPENDIX E

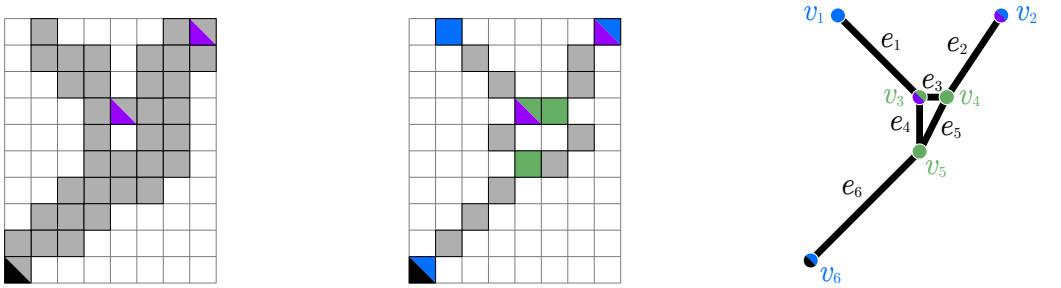
Frame extraction examples

Extracting graph models from given skeletons consists of four steps, which are type identification, graph construction, edge collapse and pruning. All these steps are used to extract graph model from the skeleton in Figure 4.1 back in Section 4.1. Here two more examples are given for clarity.

E.1 Graph $\mathcal{G}_1$

Graph $\mathcal{G}_1$ with all node types identified comes from the voxel chain shown in Figure E.1b. The mesh leading to $\mathcal{G}_1$ is identical to the mesh in Figure 4.1b, but in $\mathcal{G}_1$, one of the joint node is tagged.

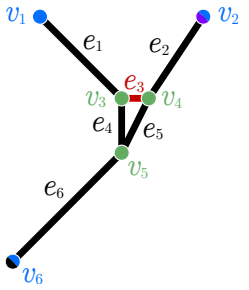
As can be seen in Figure E.2, the edge $e_3 \in \mathcal{G}_1$, connected to the tagged joint node $v_3 \in \mathcal{G}_1$, has weight of 2. Using Algorithm 4.4 to remove short edge $e_3 \in \mathcal{G}_1$ yields a slightly different result from deleting $e_3 \in \mathcal{G}_1$, because the merged node after the removal acquires the coordinate and tag of the tagged node. $e_3 \in \mathcal{G}_1$ is removed by



(a) Voxel model (b) Voxel chain from (a) (c) Graph $\mathcal{G}_1$ from (b)

Fig. E.1 $\mathcal{G}_1$: graph extracted from the voxel chain

merging row v_4 to row v_3 then crossing column e_3 and row v_4 in $\boldsymbol{B}(\mathcal{G}_1)$, as can be seen in Figure E.2. The tagged node $v_3 \in \mathcal{G}_1$ remains its position and tag. After the

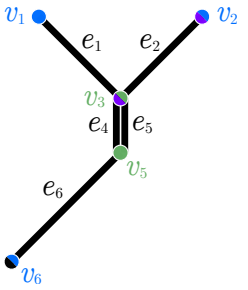


(a) $\mathcal{G}_1$

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	4	0	0	0	0	0
v_2	0	4	0	0	0	0
v_3	4	4	4	3	4	0
~~v_4~~	~~0~~	~~4~~	~~2~~	~~0~~	~~4~~	~~0~~
v_5	0	0	0	3	4	5
v_6	0	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.2 $\mathcal{G}_1$: remove short edge e_3

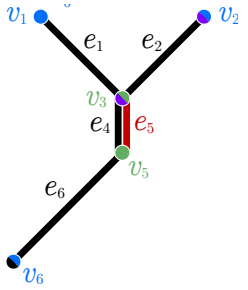


(a) $\mathcal{G}_1$

	e_1	e_2	e_4	e_5	e_6
v_1	4	0	0	0	0
v_2	0	4	0	0	0
v_3	4	4	3	4	0
v_5	0	0	3	4	5
v_6	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.3 $\mathcal{G}_1$: detecting the duplicate edges e_4 and e_5

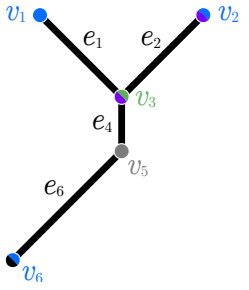


(a) $\mathcal{G}_1$

	e_1	e_2	e_4	e_5	e_6
v_1	4	0	0	0	0
v_2	0	4	0	0	0
v_3	4	4	3	4	0
v_5	0	0	3	4	5
v_6	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.4 $\mathcal{G}_1$: delete edge e_5

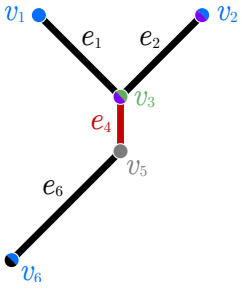


(a) $\mathcal{G}_1$

	e_1	e_2	e_4	e_6
v_1	4	0	0	0
v_2	0	4	0	0
v_3	4	4	3	0
v_5	0	0	3	5
v_6	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.5 $\mathcal{G}_1$: node v_5 turns out to be a regular node

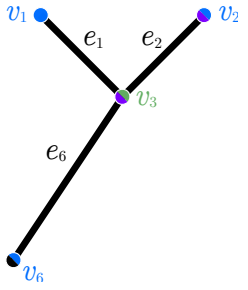


(a) $\mathcal{G}_1$

	e_1	e_2	e_4	e_6
v_1	4	0	0	0
v_2	0	4	0	0
v_3	4	4	6	5
~~v_5~~	~~0~~	~~0~~	~~3~~	~~5~~
v_6	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.6 $\mathcal{G}_1$: merge node v_5 to node v_3

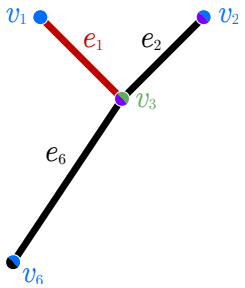


(a) $\mathcal{G}_1$

	e_1	e_2	e_6
v_1	4	0	0
v_2	0	4	0
v_3	4	4	5
v_6	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.7 $\mathcal{G}_1$: regular node v_5 is removed

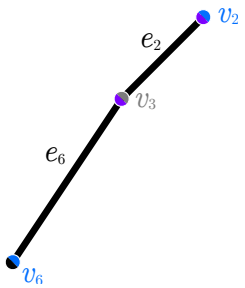


(a) $\mathcal{G}_1$

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_2 & e_6 \end{array} \\ \begin{pmatrix} \cancel{4} & \cancel{0} & \cancel{0} \\ 0 & 4 & 0 \\ 4 & 4 & 5 \\ 0 & 0 & 5 \end{pmatrix} \end{array}$$

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.8 $\mathcal{G}_1$: remove regular node v_3



(a) $\mathcal{G}_1$

$$\begin{array}{c} \\ v_2 \\ v_3 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cc} e_2 & e_6 \end{array} \\ \begin{pmatrix} 4 & 0 \\ 4 & 5 \\ 0 & 5 \end{pmatrix} \end{array}$$

(b) $\boldsymbol{B}(\mathcal{G}_1)$

Fig. E.9 $\mathcal{G}_1$: final graph

edge collapse, a duplicate edge pair $e_4 \in \mathcal{G}_1$ and $e_5 \in \mathcal{G}_1$ occurs as in Figure E.3. The duplicate edge $e_5 \in \mathcal{G}_1$ is removed in Figure E.4 by crossing column e_5 in $\boldsymbol{B}(\mathcal{G}_1)$.

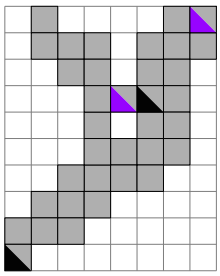
Losing edge $e_5 \in \mathcal{G}_1$, node $v_5 \in \mathcal{G}_1$ becomes untagged regular node. The function in Algorithm 4.3 will merge $v_5 \in \mathcal{G}_1$ to $v_3 \in \mathcal{G}_1$ and collapse edge $e_4 \in \mathcal{G}_1$. This can be seen in Figure E.5 and E.6. The graph in Figure E.7 is free of untagged regular nodes or duplicate edges.

The last step is graph pruning. Algorithm 4.5 finds there exists one regular node $v_1 \in \mathcal{G}_1$ in Figure E.7a. Thus the zero-stress member $e_1 \in \mathcal{G}_1$ is removed by crossing column e_1 and row v_1 in $\boldsymbol{B}(\mathcal{G}_1)$, see Figure E.8.

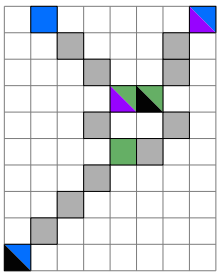
The final graph is shown in Figure E.9. $\mathcal{G}_1$ in Figure E.9 and $\mathcal{G}_1$ in Figure 4.14 come from the voxel chain with identical topology, but different positions of tags lead to difference in their final graphs. In the approach proposed in Section 4.1, the graph always alters itself in order to preserve the positions of tagged nodes given by the user to mark the important features.

E.2 Graph $\mathcal{G}_2$

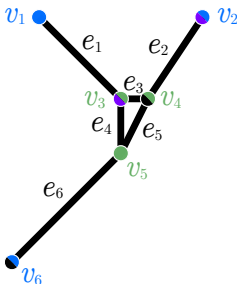
Similar to the previous example, the voxel model in Figure E.10, which is used to generate $\mathcal{G}_2$, also has the identical mesh to that in Figure 4.1. Instead, two joint voxels are tagged.



(a) Voxel model



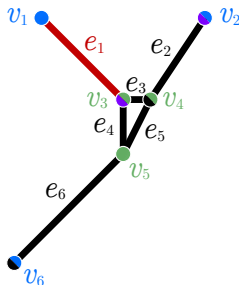
(b) Voxel chain from (a)



(c) Graph from (b)

Fig. E.10 $\mathcal{G}_2$: graph extracted from the voxel chain

Though the weight of $e_3 \in \mathcal{G}_2$ is 2, it cannot be collapsed, since both of the nodes $v_3 \in \mathcal{G}_2$ and $v_4 \in \mathcal{G}_2$ are tagged. Algorithm 4.4 skips $e_3 \in \mathcal{G}_2$ and continues to check other possible short edges in $\mathcal{G}_2$. It turns out that the graph in Figure E.10c is free of removable short edges. Next, the pruning algorithm, Algorithm 4.5, finds the zero-stress member $e_1 \in \mathcal{G}_2$ (with one regular end node $v_1 \in \mathcal{G}_2$). This zero-stress member is removed in Figure E.11 by deleting column e_1 and row v_1 in $\boldsymbol{B}(\mathcal{G}_2)$, which yields the final graph presented in Figure E.12.

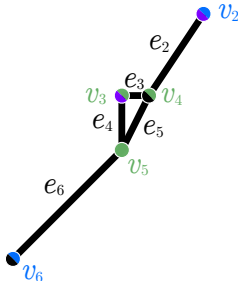


(a) $\mathcal{G}_2$

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	4	0	0	0	0	0
v_2	0	4	0	0	0	0
v_3	4	0	2	3	0	0
v_4	0	4	2	0	4	0
v_5	0	0	0	3	4	5
v_6	0	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_2)$

Fig. E.11 $\mathcal{G}_2$: zero-stress member e_1 is to be deleted



(a) $\mathcal{G}_2$

	e_2	e_3	e_4	e_5	e_6
v_2	4	0	0	0	0
v_3	0	2	3	0	0
v_4	4	2	0	4	0
v_5	0	0	3	4	5
v_6	0	0	0	0	5

(b) $\boldsymbol{B}(\mathcal{G}_2)$

Fig. E.12 $\mathcal{G}_2$: final graph